吴吉义 王中友等编著

程序员突击

- 由浅入深、循序渐进，真正**全面掌握**系统开发技术。
- 以真实的**项目开发**与实现为主线，对各个设计元素进行深入讲解。
- 精选多个**典型案例**，并结合基础知识对系统进行分析、设计与编程实现。
- 对**主流框架**进行详细分析并整合至真实项目中。
- 强大的**技术支持**，读者可在学赛网社区“书评在线”版块中与作者进行直接交流。
- 丰富的**程序资源**，读者也可以在希赛网下载中心或学赛网下载中心下载本书所有实例的源代码和实战项目的源代码。

MySQL原理与Web系统开发

程序员突击系列



清华大学出版社

程序员实击——MySQL 原理与 Web 系统开发

吴吉义　王中友　等编著

清华大学出版社

北　京

内 容 简 介

本书分为 4 篇共 13 章，包括步入 MySQL 开发的殿堂、苦练基本功、突出重围 项目实战和高级开发技术。本书使用的开发环境是 JDK 1.5+Tomcat 5.5+ Eclipse 3.1+ MySQL 5.1/Oracle 8i，逐步引领读者从基础到各个知识点的学习，然后开发出完整的系统。全书内容由浅入深，辅以大量的实例说明，并给出了 4 个完整的项目案例，且 4 个项目案例均遵循大中型软件企业规范的程序设计。

本书供有一定 Java Web 编程基础的程序员作为参考用书，也可供社会 Java 技术培训班作为教材使用，对于缺乏项目实战经验的程序员来说可用于快速积累项目开发经验。

图书在版编目（CIP）数据

程序员突击——MySQL 原理与 Web 系统开发/吴吉义，王中友等编著. —北京：清华大学出版社，2009.8

ISBN 978-7-302-18466-9

I. 程… II. ①吴… ②王… III. ①关系数据库-数据库管理系统，MySQL ②主页制作-程序设计 IV. TP311.138 TP393.092

中国版本图书馆 CIP 数据核字（2008）第 132828 号

责任编辑：涂 荣 张丽萍
封面设计：刘 超
版式设计：王世情
责任校对：王 云
责任印制：
出版发行：清华大学出版社 **地 址：**北京清华大学学研大厦 A 座
http://www.tup.com.cn **邮 编：**100084
社 总 机：010-62770175 **邮 购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经 销：全国新华书店
开 本：185×260 **印 张：**24.5 **字 数：**563 千字
版 次：2009 年 8 月第 1 版 **印 次：**2009 年 8 月第 1 次印刷
印 数：1～4000
定 价：39.80 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103 产品编号：030409-01

前　　言

MySQL 是完全网络化的跨平台关系型数据库系统，同时是具有客户机/服务器（Client/Server，C/S）体系结构的分布式数据库管理系统。它具有功能强、使用简便、管理方便、运行速度快、安全可靠性强等优点，用户可利用许多语言编写访问 MySQL 数据库的程序。另外，MySQL 在 UNIX、Linux 等操作系统下是免费的，在 Windows 操作系统下，可免费使用其客户机程序和客户机程序库。

基于 MySQL 的 J2EE 浏览器/服务器（Browser/Server，B/S）系统架构，是开发中小型信息系统较理想的选择。

一、章节内容介绍

全书分为 4 篇共 13 章。

❑ 第 1 篇“步入 MySQL 开发的殿堂”分为 3 章。

第 1 章对数据库技术的发展概况作了简要的说明，同时介绍一些常用的小型数据库和大型数据库，着重介绍 MySQL 数据库，MySQL 最常用的两种存储引擎 MyISAM 和 InnoDB。

第 2 章主要介绍 MySQL 的基础知识，包括下载、安装 MySQL 的方法、MySQL GUI 工具、MySQL 命令行客户端、MySQL 服务端的使用方法和一些常用 MySQL 实用工具程序。

第 3 章详细介绍关系数据库的标准语言 SQL，包括数据定义语言、数据查询语言、数据操纵语言、数据控制语言和嵌入式 SQL 等。

❑ 第 2 篇“若练基本功”分为 4 章。

第 4 章介绍数据分析与设计、数据库设计技巧、Power Designer 10 工作环境，最后介绍 Power Designer 10 中的正向工程与逆向工程。

第 5 章介绍 MySQL 存储过程实现细节并提供一些存储过程应用示例，简单介绍 MySQL 触发器。

第 6 章介绍 JDBC 的基本概念，对传统的 ODBC 接口的体系结构以及数据源的配置方法进行介绍，阐述了 JDBC 与 ODBC 的异同，对 JDBC API 进行详尽的介绍。

第 7 章结合大量程序代码，从实际应用的角度阐述 Connector/J 的相关知识。包括 Connector/J 的安装，如何进行 JDBC 编程，接着使用 Eclipse 工具结合运用 Struts、Hibernate 开源框架示例了一个符合 J2EE 规范的 Web 项目。

❑ 第 3 篇“突出重围　项目实战”分为 4 章。

第 8 章通过一个“用户管理系统”项目的设计与开发，描述了在 Web 中间件 Tomcat 环境下如何设计一个比较通用的用户与权限管理系统。

第 9 章通过一个“CASE 支撑系统”项目的设计与开发，实现了在网上进行故障的申告、

跟踪、处理等基本功能。描述了在 Web 中间件 Tomcat 环境下，采用 J2EE 技术和 B/S 结构，如何一步一步实现用户所要求的功能和性能。

第 10 章通过一个“文件管理系统”项目的设计与开发，描述了在 Web 中间件 Tomcat 环境下，如何设计一个比较实用的文件管理系统。

第 11 章介绍基于 Struts 和 Hibernate 实现的“教务管理系统”。本系统的开发并没有单纯地使用 JSP+Servlet 进行开发，而是结合了 Struts 和 Hibernate，这是为了使系统的结构更加清晰，同时简化开发工作。

❑ 第 4 篇“高级开发技术”分为两章。

第 12 章介绍 MySQL 5.0 高级特性，包括新 SQL 语句和 Loops 循环语句，数据导入导出工具 mysqlimport 的语法和常用选项介绍，最后还介绍了 MySQL 进行性能优化关键系统参数。

第 13 章介绍可扩展标记语言 XML 和扩展样式表语言 XSLT 的语法、使用和应用，还介绍了 MySQL 中的两个用于处理 XML 的新函数 ExtractValue()和 UpdateXML()。

项目实战案例篇的案例均以开发与实现为主线，从系统需求分析、系统总体架构的设计、数据库设计、系统目录设计、系统的关键技术、系统的各个模块的详细实现这些方面逐步深入分析，较为明晰地讲解了这个系统是如何分析、设计与编程实现的，可综合之前所学的基础知识。

本书的知识体系结构遵循了循序渐进的原则，逐步引领读者从基础到各个知识点的学习，然后开发出完整的基于 MySQL 的 Java B/S 系统。

本书内容由浅入深，并辅以大量的实例说明。本书供有一定的 Java Web 编程基础的程序员作为参考用书，也可供社会 Java 技术培训班作为教材使用，对于缺乏项目实战经验的程序员来说可用于快速积累项目开发经验。

二、技术支持

希赛是中国领先的互联网技术和 IT 教育公司，在互联网服务、图书出版、人才培养方面，希赛始终保持 IT 业界的领先地位。希赛对国家信息化建设和软件产业化发展具有强烈的使命感，利用希赛网（www.csai.cn）强大的平台优势，加强与促进 IT 人士之间的信息交流和共享，实现 IT 价值。“希赛，影响 IT”是全体希赛人不懈努力和追求的目标！

希赛网以希赛顾问团为技术依托，是中国最大的 IT 资源平台。希赛 IT 教育研发中心是希赛公司属下的一个专门从事 IT 教育、教育产品开发、教育书籍编写的部门，在 IT 教育方面具有极高的权威性。在国家权威机构发布的《计算机图书出版市场综述》中，称赞希赛丛书为读者所称道，希赛的图书已经形成品牌，在读者心目中具有良好的形象。

书中所有程序均经过了作者精心的调试。对于本书涉及的 Struts、Hibernate 等技术内容，限于篇幅，建议读者参考相关书籍。

本书由吴吉义、王中友、葛一鸣、张伟龙、王平明、龚一峰等 6 位系统分析师合作编写，最后由吴吉义博士负责完成统稿工作。参与本书编写工作的还有周泉、周进、顿海丽、张爱民、王勇、唐强、谢顺、王永明、左南、张友生、邓子云、黄婧、梁赛、杨花、彭欢等。

本书的出版得到了清华大学出版社的大力支持，在此表示感谢。也对本书“案例篇”涉及的项目建设单位和案例提供单位表示感谢。感谢本书编委会全体成员，特别感谢王中友高级工程师在本书编写工作中所做的大量工作。

由于时间仓促和作者水平有限，书中的错误和不妥之处在所难免，敬请广大读者批评指正。

有关本书的意见反馈和咨询，读者可在学赛网社区（http://bbs.educity.cn）“书评在线”版块中与作者进行交流。读者也可以在希赛网下载中心（http://data.csai.cn）或学赛网下载中心（http://data.educity.cn）下载本书所有实例的源代码和实战项目的源代码。

目　　录

第 1 篇　步入 MySQL 开发的殿堂

第 2 篇　苦练基本功

第 3 篇　突出重围　项目实战

第 4 篇　高级开发技术

第 1 篇



MySQL 这种开源的轻量级关系型数据库正吸引着越来越多的程序员和数据库管理员，使用它来构造各种各样的信息系统。

本篇内容包括 MySQL 导论、MySQL 基本操作、SQL 基础知识等内容。本篇是为初学者准备的，用于理清数据库的一些基本概念，认识 MySQL 并学会简单的操作。

第 1 章　MySQL 导论

本章对数据库技术的发展概况作了简要的说明，同时介绍一些常用的小型数据库和大型数据库，重点介绍 MySQL 数据库。通过本章的阅读，读者将对 MySQL 的发展情况、特点及其应用领域有一定的了解。1.3 节阐述 MySQL 的插件式体系结构，简单说明各种存储引擎的特点及选择方法和技巧。1.4 节重点介绍 MySQL 最常用的两种存储引擎，即 MyISAM 和 InnoDB。

1.1　数据库概述

在学习 MySQL 之前，首先简单介绍一下数据库的基本概念。

举个简单的例子来说明：每个人都有很多亲戚和朋友，为了保持与他们联系，我们常常用一个笔记本将他们的姓名、地址、电话等信息都记录下来，这样要查谁的电话或地址就很方便。这个“通讯录”就是一个最简单的“数据库”，每个人的姓名、地址、电话、邮箱等信息就是这个数据库中的“数据”。我们可以在笔记本这个“数据库”中添加新朋友的个人信息，也可以由于某个朋友的电话变动而修改他的电话号码这个“数据”。总而言之，我们使用笔记本这个“数据库”是为了能随时查到某位亲戚或朋友的地址、邮编或电话号码这些“数据”。当我们的亲戚朋友不多时，也许可以很快地从笔记本中找到所需的数据，但是当笔记本中的数据很多时，也许就要花费不少时间去查找某个朋友的联系方式了。而将这个笔记本数字化，也就是说将它的内容录入到计算机中，例如，存放在 MySQL 中，那么，即便我们有数以万计的联系人，我们也可以在一瞬间找到他们。这就是数据库的由来和作用。

随着信息产业的发展，数据库在社会中发挥了越来越重要的作用。可以说，几乎所有的信息系统都依赖于数据库。通俗地讲，数据库就是存放数据的仓库，而这个仓库是存放在计算机存储设备上，而且数据是按照一定格式存放的。按照数据库理论的定义，数据库是长期存储在计算机内的、有组织的、可共享的数据集合。在今天，电子商务、电子政务等都得到了迅速的发展，并由此产生了大量的数据。为了能够高效、准确地处理分析这些数据，人们便使用了数据库。

为了将数据存储在数据库中，通常将描述事物特征的若干个数据组成一个数据记录（Record）。例如，通讯录中的联系人，可以写成如下形式：

联系人（姓名，地址，电话，邮箱）

并将其称之为记录型，也就是数据的逻辑结构。它是对联系人这一事务的抽象描述。其中，“联系人”也称为记录名，通常在关系数据库中，也作为基本表的表名。姓名、地

址、电话、邮箱等称为字段（field），在关系数据库中，也就是各个基本表的表项。对于每个字段可以赋予特定的值，例如：

（张三，仓基社区，0571-8888888，sz@263.com）

这就成为了一条记录。在关系数据库中，若干条记录便构成一张表。

数据库带来的最直接的好处，就是实现了数据独立性。所谓数据独立性，是指数据与用户应用程序之间的独立性，也就是实现了应用程序与数据的分离。对于大多数应用程序而言，如一个电子商务网站，它必然需要有后台数据的支持才能运作。然而，这些后台数据是以什么方式存放在物理磁盘上，网站应用程序并不用关心，甚至当数据库的逻辑结构发生变化时，如数据库中原来的记录型是商品（商品编号，商品名称，价格），更改为商品（商品编号，商品名称，价格，数量），原先的网站应用程序也不用更改。前者称之为数据的物理独立性，后者叫做数据的逻辑独立性。

【特别提示】数据独立性对于系统维护而言相当重要，数据库将数据以及程序进行了分离，当数据存储方式，或者逻辑结构有改动时，应用程序可以保持不变。如果没有数据独立性，程序和数据将会高度耦合，对于系统维护而言，就是一场“灾难”。

1.1.1　数据库技术的发展

在数据库系统产生之前，人们对计算机数据的管理经历了人工管理阶段（20 世纪 50 年代中期以前）和文件系统管理阶段（20 世纪 50 年代后到 60 年代中期）。

在人工管理阶段，人们对数据的处理能力很低。从硬件上看，计算机内存小，计算速度低；从软件上看，没有操作系统的支持，更没有数据库管理软件。因此，在这个阶段，数据总量不大，数据不能长期保存，数据与应用程序不隔离，应用程序需要随着数据存储方式的变化而变化。

在文件系统阶段，计算机的存储器增大，计算速度大大提高，并且配备了操作系统。在这个阶段，数据可以长期存放，并采用文件系统管理数据。但是，使用文件系统管理数据存在许多缺点，数据冗余度大，逻辑独立性差。

1964 年，美国通用电器公司的 Bachman 等人成功开发了世界上第一个 DBMS（Database Management System）——IDS 系统，标志着人们对数据的管理进入了数据库系统阶段。与文件系统相比，数据库系统实现了数据的整体结构化。在文件系统中，虽然存在记录内的结构性，但整体上数据是无结构的，即不同文件之间的记录是没有联系的。但在数据库系统中，不仅存在记录内部的联系，而且还描述了数据之间的联系，实现了数据的整体结构化。这是数据库系统与文件系统的本质差别。同时，数据库系统使数据面向整个应用系统，降低了数据的冗余度，实现了数据的共享。

体现数据库整体结构性的典型示例就是外键约束。如图 1-1 所示，在订单记录中的收货人编号必须存在于描述收货人信息的特定的记录收货人中；否则，这便是一个非法的收货人编号。在数据库中，不同记录或表之间的这种联系，便使数据库数据整体结构化。

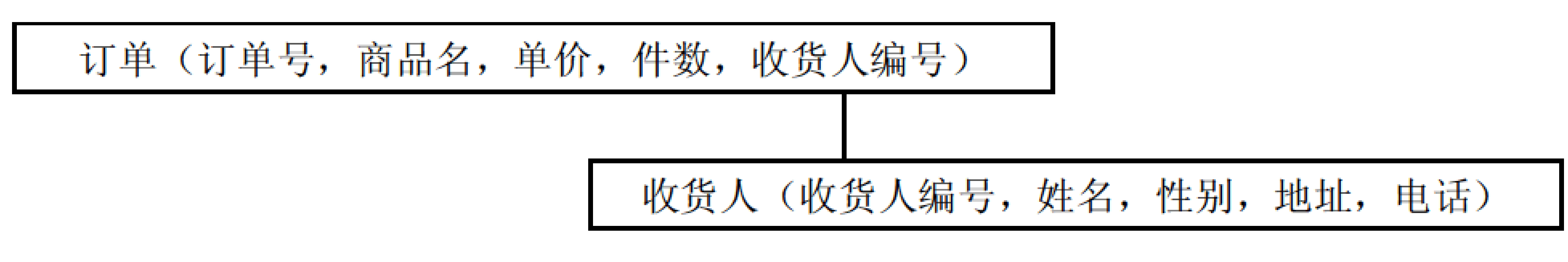


图 1-1　外键约束

数据库系统的发展经历了以下几个阶段。

第一代数据库技术以层次数据库和网状数据库为代表。其主要特点是支持三级模式结构；用指针来表示数据之间的联系；数据定义语言（Data Definaition Language，DDL）和数据操纵语言（Data Manipulation Language，DML）相对独立；数据库采用过程性（导航式）语言，用户在操作数据库时不但说明要做什么，还要说明怎么做。例如，在查找语句中不但要说明查找的对象，而且要规定存取路径。这和现在被广泛使用的非过程性语言 SQL 是不同的。

第二代数据库技术，也就是现在被广泛应用的关系数据库系统。MySQL 正是属于这类数据库。关系数据库有严格的数学理论作为基础，概念清晰，易于使用。1970 年，美国 IBM 公司 San Jose 研究实验室研究员 E.F.Codd 提出了关系数据库模型，奠定了关系数据库理论的基础。E.F.Codd 也因此获得了 1981 年的计算机图灵奖。1974 年， San Jose 实验室研制成功 System R，这是世界上最早的关系数据库管理系统（RDBMS）。1980 年后，众多 RDBMS 产品相继推出，包括 Oracle、Infomix、Sybase 等。1990 年后，RDBMS 不断发展，能够支持分布式数据库、开放环境下异构数据库互连，OLTP（On-Line Transaction Processing）联机事务处理和 OLAP（On-Line Analytical Processing）联机分析处理。

第三代数据库技术，以面向对象数据库为代表。这一代数据库管理系统基于扩展的关系数据库模型或者面向对象数据库模型，目前，该技术尚未完全成熟。但是它支持包括数据、对象的管理，能够很好地和面向对象设计技术相融合。因此，许多商品化的关系数据库管理系统也都进行了扩充，增加了面向对象特性，发展成 ORDBMS（Object-Relation DBMS）。

1.1.2　数据模型

模型是对现实事物的一种抽象。它是人们为了更好地研究现实事物而建立的一种对现实事物的模拟。它客观地表现了现实事物的特征。因为计算机只能存储数据，而不能存储处理现实事物。因此，必须先将现实事物转化为数据模型，才能交给计算机处理。例如，在网上购物时，计算机需要处理用户订单。但是计算机并无法直接处理一张订单，因此必须先将订单转化为计算机能够存储和处理的数据。而这些数据结构，体现了订单的数据特征，就是一种数据模型。数据库中的数据模型一般包括数据结构、数据操作和完整性约束 3 个要素。例如，订单（订单号，商品名，单价，件数，收货人编号）就是订单的一种数据结构。因为用户可以有下订单、撤销订单等操作，这些操作可能会引起对订单数据的插入、修改和删除。有时还会有查询等操作，这就是数据操作。而订单的单价、件数不可能为负数，这就约束了两者的取值范围。这是一种完整性约束，称之为实体完整性。假设此时有

数据结构：收货人（收货人编号，姓名，性别，地址，电话）。显然，在订单（订单号，商品名，单价，件数，收货人编号）中的收货人编号必须存在收货人（收货人编号，姓名，性别，地址，电话）之中，否则订单中的收货人编号就毫无意义。这种完整性约束，称之为参照完整性。

1. 概念数据模型

概念数据模型（Conceptual Data Model）是最上层的数据模型。它与具体的数据库类型无关，是数据库设计人员和用户交流的语言。概念数据模型十分接近现实世界，应用于数据库设计的初始阶段。最常见而且被广泛应用的概念模型是实体-联系（E-R）模型，简称E-R 模型。

2. 结构数据模型

目前，结构数据模型中最常用的结构数据模型有 4 种。分别是：层次模型、网状模型、关系模型和面向对象模型。其中，层次模型和网状模型统称为非关系模型。它们的数据结构可以和图论中的有向图相对应，比较直观。

- 层次模型

在现实生活中，许多实体之间的联系就是一种自然的层次关系。例如，一个文件系统的目录结构，就是一个很好的例子。

层次模型中，每个结点只有一个双亲结点。整个模型中，也只有根结点没有双亲结点。每个结点都表示一个实体型（集）。实体之间的联系用结点之间的连线表示。这种联系是双亲结点与子女结点之间的一对多联系。因此，层次模型表达一对一、一对多的联系是非常直观的。但是，对于表达多对多联系时，层次模型需要通过辅助手段才能表现，显得笨拙、复杂。

- 网状模型

层次模型可以方便地表示现实世界中的层次关系。而实际上，现实世界中存在着大量的非层次关系。对于这些关系，采用层次模型是很不直接的。而网状模型可以克服这个缺点。

例如，用户的权限分配。同一用户可以拥有多种不同的权限，而同一权限可以被系统的多个不同用户同时拥有，这就是多对多的联系。图 1-2 显示了用户与权限的多对多关系。

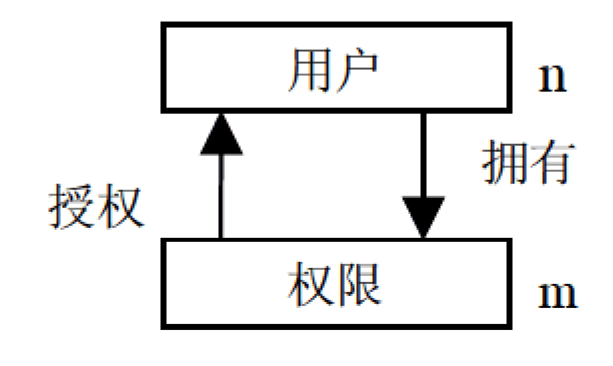


图 1-2　多对多关系

- 关系模型

关系模型是现在最重要的一种数据模型。自 20 世纪 80、90 年代以来，软件厂商新推出的数据库管理系统几乎都是以关系模型为基础的。现在，包括 MySQL 在内的多种数据库都是基于关系模型的。

关系模型的逻辑结构是一张二维表，简称表，也可称之为关系。它由行和列组成。二维表中的一列也可以称之为属性。如表 1-1 所示，订单表有 5 列，其属性分别是订单号、商品名、单价、件数、收货人编号。二维表的表头，即（订单号，商品名，单价，件数，收货人编号）对应一个关系模式。表 1-1 所对应的关系模式可以描述为：订单（订单号，商品名，单价，件数，收货人编号）。在二维表中，除了表头以外的任意非空行为一个元组，即一条记录。如（0001，专家导学基于 MySQL 的 Java B/S 系统开发，50，1，000325）就是一个元组。

表 1-1　订单表

订　单　号	商　品　名	单　　价	件　　数	收货人编号
0001	专家导学基于 MySQL 的 Java B/S 系统开发	50	1	000325
0002	专家导学 Tomcat 应用开发	50	2	000333

- 面向对象模型

面向对象（Object Oriented，OO）是当前计算机界关心的重点，它是 20 世纪 90 年代软件开发方法的主流。面向对象的概念和应用已超越了程序设计和软件开发，扩展到很宽的范围。对象是人们要进行研究的任何事物，从最简单的整数到复杂的飞机等均可看作对象，它不仅能表示具体的事物，还能表示抽象的规则、计划或事件。

对象模型表示了静态的、结构化的系统数据性质，描述了系统的静态结构，它是从客观世界实体的对象关系角度来描述的，表现了对象的相互关系。该模型主要关心系统中对象的结构、属性和操作，它是分析阶段 3 个模型的核心，是其他两个模型的框架。

动态模型是与时间和变化有关的系统性质。该模型描述了系统的控制结构，它表示了瞬间的、行为化的系统控制性质，它关心的是系统的控制、操作的执行顺序，它表示从对象的事件和状态的角度出发，表现了对象的相互行为。该模型描述的系统属性是触发事件、事件序列、状态、事件与状态的组织。使用状态图作为描述工具。它涉及事件、状态、操作等重要概念。

功能模型描述了系统的所有计算。功能模型指出发生了什么，动态模型确定什么时候发生，而对象模型确定发生的客体。功能模型表明一个计算如何从输入值得到输出值，它不考虑计算的次序。功能模型由多张数据流图组成。数据流图用来表示从源对象到目标对象的数据值的流向，它不包含控制信息，控制信息在动态模型中表示，同时数据流图也不表示对象中值的组织，值的组织在对象模型中表示。

3. 物理数据模型

物理数据模型是描述数据在存储介质商组织结构的数据模型，它不但与具体的 DBMS 有关，而且还与操作系统和硬件有关，是物理层次的数据模型。为了保证数据的独立性和可移植性，DBMS 完成了大部分物理数据模型的实现工作，而设计者只需关心设计索引等特殊结构即可。

1.1.3　常用数据库介绍

1. Access

Access 是 Microsoft 的产品，伴随着 Office 一起发布。作为一个单文件的数据库系统，

它使用简单，但是功能并不强大，不支持事务、触发器、存储过程等应用。不适用于对可靠性要求较高的场合。

2. HSQLDB

HSQLDB 是一个开源纯 Java 的数据库，小巧方便。具有标准的 SQL 语法和 Java 接口，支持事务处理，支持 Java 存储过程和函数。它属于 BSD 的 license，可以自由下载，并且可以安装使用在商业产品之上。

3. SQLITE

SQLITE 是开源单一文件数据库。支持 ACID 事务，触发器；零配置，不需安装，支持数据大小达 2TB。但它不支持外键，内部采用 UTF8 存储数据，对于中文数据的处理，都必须调用编码函数，略有不便。

4. SQL Server

SQL Server 是由微软开发的一款数据库管理系统。其最新版本是 SQL Server 2008，适合于中小企业使用，只能在 Windows 操作系统下运行。目前包含多种版本，如学习版、工作组版、开发版、标准版、企业版和移动版，用户可以根据实际情况选择适合自己的版本。但是除学习版外，其他版本价格不菲，其中企业版售价约为 35 万人民币，标准版售价约为 7 万人民币。

5. Oracle

Oracle 数据库是 Oracle 公司的产品，是目前最为流行的大型数据库之一，可以运行在 Windows 和类 UNIX 等多种操作系统下，适合于大型企业使用。目前最新版本是 Oracle 11g。Oracle 提供的功能是其他中小型数据库望尘莫及的。Oracle 提供了高度的可用性、可伸缩性、可管理性和安全性，支持集群应用、数据仓库、内容管理等功能。但 Oracle 数据库价格较高，各项产品的售价大约在 2 万~50 万人民币不等。而且，Oracle 数据库的学习周期较长，不易掌握。

6. DB2

DB2 是 IBM 5 个软件品牌之一，属于大型数据库，是 Oracle 的强有力的竞争对手。可以运行在多种不同的操作系统下，包括 UNIX、Windows、AS/400 和 OS/390。DB2 对商业智能、内容和记录管理、异构数据库集成有较好的支持。其价格大约在 2 万美元左右。

1.2　MySQL 简介

一开始 MySQL 是作为小型轻量级关系数据库推出的，主要定位于小型信息系统开发中的数据管理。近年来，随着其研发技术的不断进步和版本的持续升级，MySQL 在应用开发中正表现出越来越出色的稳定性和可靠性。

1.2.1 MySQL 是什么

MySQL 是最流行的开放源码 SQL 数据库管理系统，它是由 MySQL AB 公司开发、发布并支持的。MySQL AB 是由多名 MySQL 开发人创办的一家商业公司。它是一家第二代开放源码公司，结合了开放源码价值取向、方法和成功的商业模型。MySQL 的正式发音是 My Ess Que Ell（而不是 my sequel）。

MySQL 的网站（http://www.MySQL.com）上给出了关于 MySQL 的最新信息。在本书撰写时，MySQL 的最新版本为 6.0，但若无特殊说明，本书中以 MySQL 5.1 为例进行阐述。

MySQL 是一种数据库管理系统。

数据库是数据的结构化集合。它可以是任何东西，从简单的购物清单到画展，或企业网络中的海量信息。要想将数据添加到数据库，或访问、处理计算机数据库中保存的数据，需要使用数据库管理系统，如 MySQL 服务器。计算机是处理大量数据的理想工具，因此，数据库管理系统在计算机应用方面扮演着关键的角色，或是作为独立的实用工具，或是作为其他应用程序的组成部分。

MySQL 是一种关系数据库管理系统。

关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大的仓库内。这样就增加了速度并提高了灵活性。MySQL 的 SQL 指的是“结构化查询语言”。SQL 是用于访问数据库的最常用标准化语言，它是由 ANSI/ISO SQL 标准定义的。SQL 标准自 1986 年以来不断演化发展，有数种版本。

1.2.2 MySQL 的特点

- 开放源代码

“开放源码”意味着任何人都能使用和改变软件。任何人都能从 Internet 上下载 MySQL 软件，而无须支付任何费用。如果愿意，可以研究源码并进行恰当的更改，以满足自己的需求。MySQL 软件采用了 GPL（GNU 通用公共许可证），http://www.fsf.org/licenses/，定义了在不同情况下可以用软件做的事和不可做的事。如果对 GPL 不满意，或需要在商业应用程序中嵌入 MySQL 代码，也可从 MySQL AB 公司购买商业许可版本。由此可见，与一般的商用数据库相比，MySQL 不仅在价格上远远胜于它们，而且由于源代码的开发，提供了商用数据库没有的灵活性。

- 反应速度

数据库的反应速度是相当重要的。它直接决定了应用系统的性能。在 MySQL 网站公布的基准测试结果发现，MySQL 对于某些指令的执行速度，甚至比 SQL Server 和 Oracle 等商用数据库更快。MySQL 服务器已能提供丰富和有用的功能。它具有良好的连通性、速度和安全性，这使得 MySQL 很适合作为 Internet 上的数据库。

- 易用性

不像 Oracle 或者 DB2 等大型数据库系统，MySQL 的易用性相当好。初学者几乎只需

要几个小时就能掌握 MySQL 的基本知识。同时，MySQL 有一本详细的使用手册、大量的在线指南、一个广泛的开发社区和大量的书籍。因此，学习 MySQL 不会给用户带来任何不便和不快。同时，MySQL 提供了良好的 GUI 工具，使得 MySQL 的使用和管理更加方便、简捷。

❑ 多种工作模式

MySQL 数据库软件是一种客户端/服务器系统，由支持不同后端的一个多线程 SQL 服务器、数种不同的客户端程序和库、众多管理工具和广泛的应用编程接口 API 组成。MySQL 服务器还有嵌入式版本。它可以集成到任何应用程序当中。

❑ 支持多种数据类型

MySQL 支持带符号/无符号整数，1、2、3、4、8 字节长，FLOAT、DOUBLE、CHAR、VARCHAR、TEXT、BLOB、DATE、TIME、DATETIME、TIMESTAMP、YEAR、SET、ENUM 以及 OpenGIS 空间类型，定长和可变长度记录。这些数据类型可以满足几乎所有应用程序的应用。

为了优化存储，在任何情况下均应使用最精确的类型。例如，如果列的值的范围为 1~99999，若使用整数，则 MEDIUMINT UNSIGNED 是最好的类型。在所有可以表示该列值的类型中，该类型使用的存储空间最少。

❑ 可伸缩性和限制

MySQL 服务器可以处理含 5 千万条记录的数据库，甚至有些用户将 MySQL 用于含 60000 个表和约 50 亿行的数据库。

每个表可支持高达 64 条索引（在 MySQL 4.1.2 之前为 32 条）。每条索引可由 1~16 个列或列元素组成。最大索引宽度为 1000 字节（在 MySQL 4.1.2 之前为 500 字节）。索引可使用具备 CHAR、VARCHAR、BLOB 或 TEXT 列类型的列前缀。

在 MySQL 5.1 中所支持的表的大小几乎可以满足任何用户的需求。实际上，表的大小限制并不取决于 MySQL 本身，而取决于操作系统所支持的文件系统。

InnoDB 存储引擎将 InnoDB 表保存在一个表空间内，该表空间可由数个文件创建。这样，表的大小就能超过单独文件的最大容量。表空间可包括原始磁盘分区，从而使得很大的表成为可能。表空间的最大容量为 64TB。

在表 1-2 中，列出了一些关于操作系统文件大小限制的示例。

表 1-2　操作系统文件大小的限制

操 作 系 统	文件大小的限制
Linux 2.2-Intel 32-bit	2GB（LFS: 4GB）
Linux 2.4+	（using ext3 filesystem）4TB
Solaris 9/10	16TB
NetWare w/NSS filesystem	8TB
win32 w/ FAT/FAT32	2GB/4GB
win32 w/ NTFS	2TB（可能更大）
MacOS X w/ HFS+	2TB

❑　多用户支持

MySQL 是一个完全多用户系统，这意味着多个客户可以同时访问和使用一个（或更多的）MySQL 数据库。这对于网络的应用程序（要求支持由多个远程客户同时建立的连接）是具有特殊意义的。MySQL 也包括一个强有力的、灵活的权限系统，它允许管理者使用基于用户和基于主机的认证方案的组合来限制对敏感数据的访问。

❑　可移植性

MySQL 主要在 Linux（SuSE 和 Red Hat）、FreeBSD 和 Sun Solaris（版本 8 和版本 9）上开发。但是可以移植到多种操作系统。要求 MySQL 服务器支持线程，客户端则需要 C++编译器即可。MySQL 支持的系统包括 Linux、Solaria、FreeBSD、OS/l、MacOS 以及 Windows 95/98/Me/2000/XP 和 NT，它可以在一系列体系结构上运行，包括 Intel x86、Alpha、SPARC、PowaPC 和 IA64，它还支持从低档的 386 系列到高档的 Pentinum 机器和 IBM.zSeries 大型机等很多的硬件配置。

❑　遵循现有标准

MySQL 服务器能够工作在不同的 SQL 模式下，并能针对不同的客户端以不同的方式应用这些模式。这样，应用程序就能对服务器操作进行量身定制以满足自己的需求。这类模式定义了 MySQL 应支持的 SQL 语法，以及应该在数据上执行何种确认检查。这样，就能在众多不同的环境下，与其他数据库服务器一起更容易地使用 MySQL。可以使用--sql-mode="modes"选项，通过启动 MySQL 来设置默认的 SQL 模式。从 MySQL 4.1 开始，也能在启动之后，使用 SET [SESSION|GLOBAL] sql_mode='modes'语句，通过设置 sql_mode 变量更改模式。

❑　国际化

MySQL 服务器可使用多种语言向客户端提供错误消息，极大地优化了与用户的交互方式。

对数种不同字符集的全面支持，包括 gbk、gb2321、latin1（cp1252）、german、big5、ujis 等。从 MySQL 4.1 开始，提供了对 Unicode 的支持。

❑　广泛的应用程序支持

MySQL 提供面向各种编程语言的 API，如 C、C++、ODBC、Java、PHP、Perl、Python、Tcl 等的 API。因此，具有广泛的适应性。

❑　事务支持

事务是保证数据库并发性和数据一致性的重要手段之一。它是用户定义的一组操作序列的集合，是数据恢复和并发控制的基本单位。一个事务中的操作，要么全部被执行，要么全部不被执行。

MySQL 提供了事务和非事务支持的存储引擎。InnoDB 存储引擎提供了全面的 ACID 兼容性。对于那些非事务支持的存储引擎，MySQL 也提供了保证数据一致性的有效方法。

【特别提示】 所谓 ACID，即原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持续性（Durability），是事务所具备的 4 个特性。原子性指一个事务是一个不可分割的逻辑工作单位，事务中所有的操作要么全部完成，要么全都不做。

一致性指事务的执行结果必须使数据库从一个一致性状态变到另一个一致性状态。隔离性指一个事务的执行不能被其他事务干扰，即事务内部的操作及其所有的数据对并发执行的其他事务是隔离的，并发执行的各个事务互不干扰。持续性指一个事务一旦被提交，它对数据库中数据的改变是持久的，其他操作或故障不对其产生影响，即当事务被提交后，即使系统崩溃，但事务对数据库的影响依然存在而且持久。

- 外键支持

外键是保证参照完整性的重要手段之一。

InnoDB 存储引擎支持对外键约束的检查功能，这些约束包括 CASCADE、ON DELETE 和 ON UPDATE。对于 InnoDB 之外的其他存储引擎，MySQL 服务器能够解析 CREATE TABLE 语句中的 FOREIGN KEY 语法，但不能使用或保存它。

- 视图支持

MySQL 5.0 开始已经支持视图功能（包括可更新视图）。在 5.0.1 和更高版本中，提供了二进制版的视图功能。View（视图）十分有用，它允许用户像单个表那样访问一组关系（表），它提供了对外模式的支持，有利于数据库设计更好地面向终端用户。视图也能限制对行的访问（特定表的子集）。对于列控制的访问，可使用 MySQL 服务器中的高级权限系统。

- 存储过程和触发器

MySQL 5.1 中已经支持存储过程和基本的触发器功能。触发器是与表有关的命名数据库对象，当表上出现特定事件时，将激活该对象。

- 采用多处理器和多线程

为了利用多处理器体系结构，MySQL 采用多线程设计，轮询在多个处理器之间分派线程，以达到更高的并行度。根据不同的平台，MySQL 使用各种线程程序包。

- 复制

复制是一种数据发布机制，它运行在远距离的地方放置表和数据库副本，以使用户可以更加方便地访问它们。MySQL 支持单项、异步的复制。产品数据库就是使用这种复制机制的典型。例如，一个国内或全球公司可能只有一个通用的集中更新产品的数据库，但是每个本地办公室都能使用它。其方法不是每次需要访问数据库时都查询远程的表，而是为每个人发布一个副本，这样的方法会更有效，因为每个办公室只负担一次传输的开销。复制机制依赖 MySQL 日志机制来追踪对数据库的所有改变。主数据库把这个日志传送给从数据库，然后从数据库在自己的数据上应用这个日志。一个从数据库不能阻止用户在复制环境以外的情况下更新日志，因此必须确保这种情况不会发生，以确保数据库同步的确信度。

1.2.3　MySQL 的应用

对于中小型网站的后台数据库，如果不愿意选择微软的产品（与 MySQL 相比它们非常昂贵，事实上，MySQL 正是 SQL Server 的强劲对手），那么 MySQL 几乎成为唯一的选择。

可以说，MySQL 是中小型数据库的首选和主流产品。然而，MySQL 不仅局限于中小型数据库领域，由于它极好的扩展性，作为大型数据库使用，MySQL 仍然有不俗的表现，本节将简单介绍一下 MySQL 作为大型数据库使用的案例。

1．Cox 通信公司用 MySQL 构建大型数据仓库

Cox 是全美第四大有线电视提供商，为大约 630 万用户提供服务。作为财富 500 强之一，Cox 公司以其高容量、可靠宽带传送网络和优秀的客户服务著称。

为了保证优质的性能和客户服务级别，Cox 开发了一个巨大的数据仓库应用。MySQL 数据库应用于这个企业关键性系统的核心。Cox 使用两台 IBM x 系列服务器做成了双机 MySQL 服务器。用于对外提供查询和实时轮询（polling）（每天大约 10 万次）。MySQL 数据库管理各种有线 modem 信息，维护电缆固件，向用户提供实时支持接口、加速内部、用户趋势和分析。同时，该数据库集成了公司预定的轮询和用于数据挖掘以建立射频车间的永久错误识别标准。每天，Cox 通过 MySQL 从 120 多万有线 modem 中取出数据。整个数据库包含超过 3600 个表和 20 亿行数据。每两小时 MySQL 处理大约 400 万次插入操作。

Cox 使用 LAMP 软件架构，使用 Linux+Apache+MySQL+PHP+Perl 轮询应用 Perl 语言编写，同时用 Perl 收集轮询的数据并存入主 MySQL 数据库中。应用基于 Web 的前端采用 PHP，达到了报告和实时轮询的特性。

2．Los Alamos 国家实验室依靠 MySQL 管理超过 7TB 的数据

Los Alamos 国家实验室是美国能源部下属 28 个实验室之一，是新墨西哥最大的研究所。该研究实验室有 8 个数据库，包含科学文献以及相关元数据，如摘要、作者简历和参考书目等。每个数据库来自于不同的提供者，数据存成了特殊的格式，研究人员查找起来非常不方便。为解决这个问题，该实验室开发了一个负责的数据库应用 SerchPlus，包含 5500 万科学期刊文献，能在任何地方通过浏览器查询和访问。作为一个高性能的数据库，MySQL 在 Los Alamos 实验室强壮、安全和可扩展的 SerchPlus 系统中起到了关键作用。

MySQL 在项目中体现了以下优势。

- 高性能：MySQL 存储了 14 亿行数据，采用 MyISAM 存储引擎使应用响应时间非常迅速。
- 安全性：采用 MySQL 提供安全登录。
- 复制：查询图书馆将数据库在防火墙外也复制一套，这样同时能对内部人员和外部人员使用。

3．Evite 公司依靠 MySQL 传递数以百万的邀请

Evite 是全球领先的交互多媒体公司 InterActive 公司的免费在线活动计划服务。Evite 网站向 600 万用户提供服务，每个月发出 900 万份邀请函。Evite 的成功使得它的流量每年以 80%的速度增长。

以前 Evite 使用 Oracle 长达 4 年。但他们觉得 Oracle 非常昂贵，而且对没有经验的员工来说很难使用。同时，公司需要性能和扩展性好的数据库来满足日益增长的业务需求。仅用了一年，MySQL 就成为 Evite 公司 IT 架构中的关键部分。几乎所有 Evite 网站都使用

MySQL，同时它也用来作为企业级关键性的数据仓库。

MySQL 在使用中，体现了以下优势。

- 使用简单：无论是开发者还是管理人员都很容易上手，无须新增数据库专家。
- 可靠性：在一年多的时间里，MySQL 非常稳定可靠，跟上了流量增长 80%的需求。
- 支持全文本搜索：通过全文本搜索，Evite 将过滤副本的时间从 20 小时降到了两小时。

1.3　MySQL 体系结构综述

在 MySQL 5.1 中，MySQL AB 引入了新的插件式存储引擎体系结构，允许将存储引擎加载到正在运行的 MySQL 服务器中。

1.3.1　插件式存储引擎体系结构

数据库管理人员通过选择相应的存储引擎，而不是通过编码的方式来实现特定的需求。由于 MySQL 服务器体系结构在存储级别上提供了一致和简单的应用模型和 API，应用程序编程人员和 DBA 可以不再考虑所有的底层实施细节。因此，尽管不同的存储引擎具有不同的能力，应用程序与存储引擎之间是相对独立的。

在图 1-3 中，以图形方式介绍了 MySQL 插件式存储引擎体系结构。

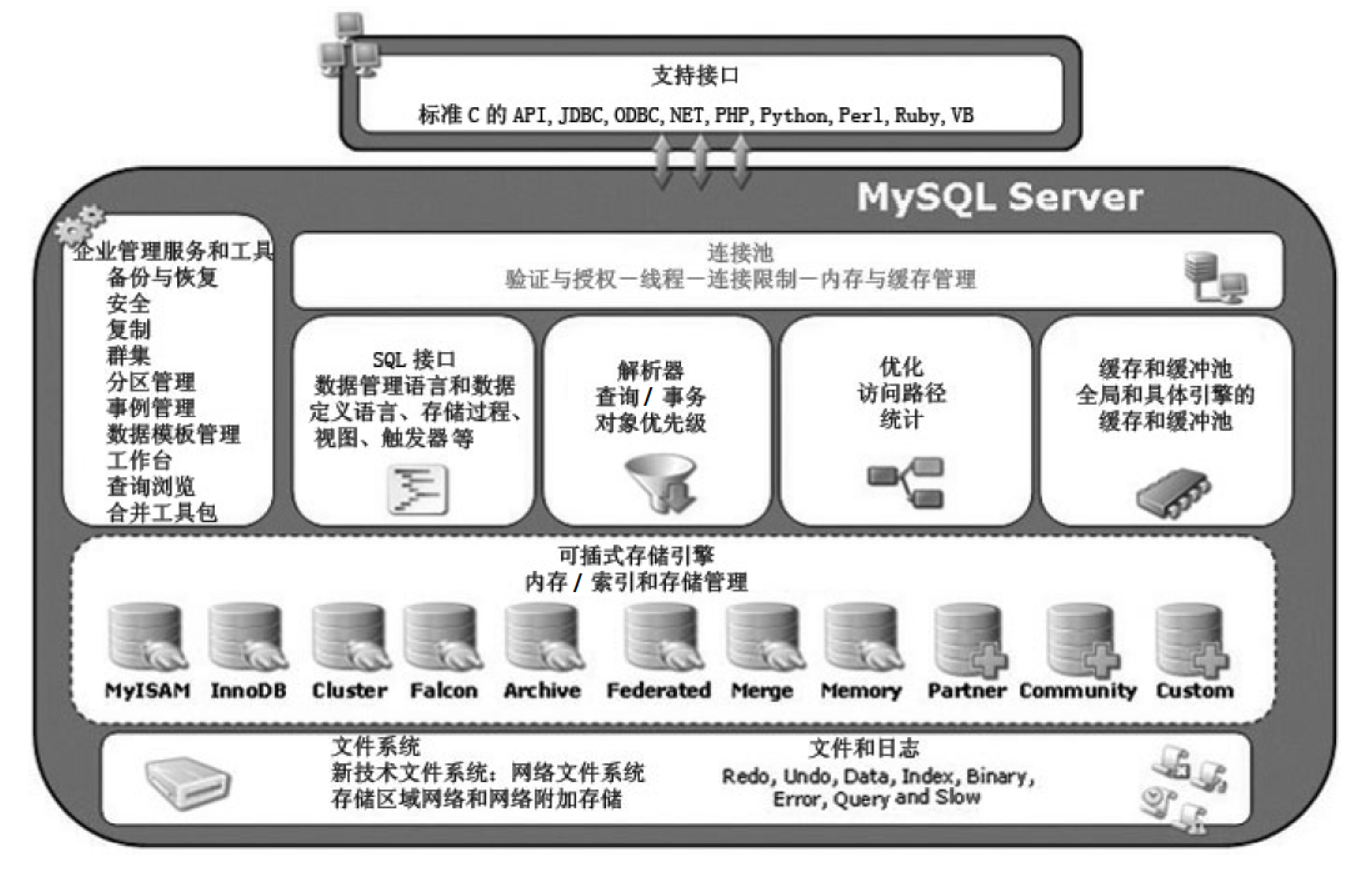


图 1-3　MySQL 插件式存储引擎体系结构

插件式存储引擎体系结构提供了标准的管理和支持服务的集合。存储引擎本身是数据库服务器的组件，负责对在物理服务器层上的基本数据进行实际的 I/O 操作。

这是一种高效的模块化体系结构，它为那些希望专注于特定应用需求的人员提供了巨大的便利，这类特殊应用需求包括数据仓库、事务处理、高可用性等，同时还能利用独立于任何存储引擎的一组接口和服务。

应用程序编程人员和 DBA 通过位于存储引擎之上的 API 和服务层来使用 MySQL 数据库。因为不是直接操作各种存储引擎，所以即使因为应用程序需求而改变了底层的存储引擎，或需要增加一个或多个额外的存储引擎，并不需要进行大的编码或者额外的操作。MySQL 服务器体系结构提供了统一和易于使用的 API，这类 API 适用于多种存储引擎，通过这种方式，该结构将应用程序与存储引擎的底层复杂性隔离开来。

1.3.2　公共 MySQL 数据库服务器层

MySQL 插件式存储引擎是 MySQL 数据库服务器中的组件，负责为数据库执行实际的数据 I/O 操作，并且提供一些特定的应用（例如视图、外键、事务支持等）。使用特殊存储引擎的主要优点之一在于，为了实现每种特殊的应用仅需提供特殊应用所需的特性，因此，数据库中的系统开销较小，最终结果具有更有效和更高的数据库性能。这也是 MySQL 被始终视为具有高性能的原因之一，在行业标准基准方面，它能匹敌或击败专有的整体式数据库。

从技术角度上看，在不同的存储引擎中，各种存储引擎所提供的不同服务主要有以下几个方面的区别。

- 并发性：某些应用程序比其他应用程序具有更多的颗粒级锁定要求（如行级锁定）。选择正确的锁定策略能够减少开销，并有助于整体性能的提升。它还包括对多种能力的支持，如多版本并发性控制（MVCC）或“快照”读取等。
- 事务支持：并非所有的应用程序都需要事务，但对的确需要事务的应用程序来说，可以选择支持事务的存储引擎，例如 InnoDB 引擎。
- 参照完整性：通过 DDL 定义的外键，服务器需要强制保持关系数据库的引用完整性。
- 物理存储：包括各种各样的物理格式，从表和索引的总的页大小、存储数据所需的格式到物理磁盘。
- 索引支持：不同的应用程序倾向于采用不同的索引策略，每种存储引擎通常有自己的编制索引方法，但某些索引方法（如 B-tree 索引）对几乎所有的存储引擎来说是相同的。
- 内存高速缓冲：不同的应用程序往往需要不同的缓存区大小和分配策略，尽管某些内存高速缓冲对所有存储引擎来说是相同的（如用于用户连接的高速缓冲、MySQL 的高速查询高速缓冲等），而各种存储引擎不同的内存高速缓冲管理方式可以根据应用程序需要进行选择。
- 性能：包括针对并行操作的多 I/O 线程、线程并发性、数据库检查点、成批插入处理等。

每个存储引擎都是为了实现响应特定的需求而设计的。因此，在设计数据库时，应该尽量避免使用我们并不需要的特性，选择合适的存储引擎。这样可以减少服务器额外的负担，提高服务器的效率。例如，在一个根本不需要事务的系统中，就应该避免选用支持事务的存储引擎。这也就是 MySQL 插件式存储引擎结构比使用专门的存储引擎结构优越的地方。

【特别提示】对于整个服务器或方案，并不一定要使用相同的存储引擎，可以为方案中的每个表使用不同的存储引擎，以满足该表特定的需求，这点很重要。

1.4　MySQL 引擎

在默认情况下，MySQL 支持 3 个引擎：ISAM、 MyISAM 和 HEAP。另外两种类型，即 InnoDB 和 Berkley（BDB）也常使用。用户能用的数据库引擎取决于 MySQL 在安装时是如何被编译的。要添加一个新的引擎，就必须重新编译 MySQL。

1.4.1　选择存储引擎

MySQL 提供的各种存储引擎在设计时考虑了不同的使用情况。为了更有效地使用插件式存储体系结构，最好了解各种存储引擎的优点和缺点。

在图 1-4 的表格中，概要展示了与 MySQL 提供的存储引擎。

Feature	MyISAM	BDB	Memory	InnoDB	Archive	NDB
Storage Limits	No	No	Yes	64TB	No	Yes
Transactions (commit, rollback, etc.)		✓		✓		
Locking granularity	Table	Page	Table	Row	Row	Row
MVCC/Snapshot Read				✓	✓	✓
Geospatial support	✓					
B-Tree indexes	✓	✓	✓	✓		✓
Hash indexes			✓	✓		✓
Full text search index	✓					
Clustered index				✓		
Data Caches			✓	✓		✓
Index Caches	✓		✓	✓		✓
Compressed data	✓				✓	
Encrypted data (via function)	✓	✓	✓	✓	✓	✓
Storage cost (space used)	Low	Low	N/A	High	Very Low	Low
Memory cost	Low	Low	Medium	High	Low	High
Bulk Insert Speed	High	High	High	Low	Very High	High
Cluster database support						✓
Replication support	✓	✓	✓	✓	✓	✓
Foreign key support				✓		
Backup/Point-in-time recovery	✓	✓	✓	✓	✓	✓
Query cache support	✓	✓	✓	✓	✓	✓
Update Statistics for Data Dictionary	✓	✓	✓	✓	✓	✓

图 1-4　MySQL 存储引擎概要

下面简单介绍常用的存储引擎。

- MyISAM：默认的 MySQL 插件式存储引擎，它是在 Web、数据仓储和其他应用环境下最常使用的存储引擎之一。通过设置 STORAGE_ENGINE 配置变量，能够更

改 MySQL 服务器的默认存储引擎。

- InnoDB：用于事务处理应用程序，具有众多特性，包括 ACID 事务支持。
- BDB：可替代 InnoDB 的事务引擎，支持 COMMIT、ROLLBACK 和其他事务特性。
- Memory：将所有数据保存在 RAM 中，在需要快速查找引用和其他类似数据的环境下，可提供极快的访问。
- Merge：允许 MySQL DBA 或开发人员将一系列等同的 MyISAM 表以逻辑方式组合在一起，并作为一个对象引用它们。适合于诸如数据仓储等 VLDB 环境。
- Archive：为大量很少引用的历史、归档或安全审计信息的存储和检索提供了完美的解决方案。
- Federated：能够将多个分离的 MySQL 服务器链接起来，从多个物理服务器创建一个逻辑数据库。十分适合于分布式环境或数据集市的应用。
- Cluster/NDB：MySQL 的簇式数据库引擎，尤其适合于具有高性能查找要求的应用程序，这类查找需求还要求具有最高的正常工作时间和可用性。
- Other：其他存储引擎包括 CSV（引用由逗号隔开的用作数据库表的文件），Blackhole（用于临时禁止对数据库的应用程序输入），以及 Example 引擎（可为快速创建定制的插件式存储引擎提供帮助）。

特别要注意，下列存储引擎提供了事务支持。

- InnoDB：通过 MVCC 支持事务，允许 COMMIT、ROLLBACK 和保存点。
- NDB：通过 MVCC 支持事务，允许 COMMIT 和 ROLLBACK。
- BDB：支持事务，允许 COMMIT 和 ROLLBACK。

【特别提示】 MVCC（Multiversion Concurrency Control）是指多版本并发控制，是一种数据库并发控制的手段。当检索数据库时，每个事务都看到一个数据的一段时间前的快照（一个数据库版本），而不管正在处理的数据当前的状态。这样，如果对每个数据库会话进行事务隔离，即可避免一个事务看到因为其他并行的事务更新同一行数据而导致的不一致的数据。使用 MVCC 多版本并发控制比锁模型的主要优点是在 MVCC 中，对检索（读）数据的锁要求与写数据的锁要求不冲突，所以读不会阻塞写，而写也从不阻塞读。

1.4.2　使用存储引擎

可以在创建新表时指定存储引擎，或通过使用 ALTER TABLE 语句指定存储引擎。 要想在创建表时指定存储引擎，可以使用 ENGINE 参数：

```
CREATE TABLE engineTest(
id INT
) ENGINE = MyISAM;
```

要想更改已有表的存储引擎，可使用 ALTER TABLE 语句：

```
ALTER TABLE engineTest ENGINE = ARCHIVE;
```

1.4.3　MyISAM 存储引擎

1．MyISAM 的特点

MyISAM 是默认存储引擎。它基于更老的 ISAM 引擎，但提供了许多扩展功能（注意 MySQL 5.1 不支持 ISAM）。

每个 MyISAM 在磁盘上存储成 3 个文件。每个文件以表名命名，扩展名指明了文件的类型。.frm 文件存储表定义。数据文件的扩展名为.MYD（MYData）。索引文件的扩展名为.MYI（MYIndex）。

要明确指定使用一个 MyISAM 表格，可以使用 ENGINE 表选项：

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

老版本的 MySQL 使用 TYPE 而不是 ENGINE（例如，TYPE = MYISAM）。MySQL 5.1 为向下兼容而支持这个语法，但推荐使用新的关键字 ENGINE。

除非默认存储引擎被改变，否则不需要显示声明 MyISAM 引擎。

MyISAM 存储引擎有以下一些特征：

（1）所有数据值先存储低字节。这使得数据库和操作系统分离。二进制兼容的唯一要求是机器使用补码和 IEEE 浮点格式（现代的大部分计算机都支持这两种格式）。

（2）先存储数据低字节并不会严重地影响速度；数据行中的字节一般是没有对齐的，顺序读取一个没有对齐的字节与反向读取这个字节所占用几乎相同的资源。而为了取得更好的索引压缩，所有的键值都先存储高字节。

（3）如果文件系统和操作系统支持大文件，那么 MyISAM 也可以支持这些大文件。

（4）一个 MyISAM 表最多可以有 2^{32} 行。如果希望 MyISAM 支持更大的表，可以使用 --with-big-tables 选项，那么表的大小可达（2^{32}）^{2} 行。

（5）每张 MyISAM 表支持最多 64 个索引。每个索引最多能有 16 列。

（6）最大的键长度是 1000 字节。如果键长度超过 250 字节，那么会使用 1024 字节的存储块去存放这个键。

（7）可以在 BLOB 和 TEXT 列上建立索引。

（8）在索引的列中允许出现 NULL 值，每个键使用 0～1 个字节。

（9）可以把数据文件和索引文件放在不同目录，使用 DATA DIRECTORY 和 INDEX DIRECTORY 选项 CREATE TABLE 可以获得更高的速度。

（10）每个字符列可以支持不同的字符集。

（11）在 MyISAM 索引文件中有一个标志，它表明表是否被正确关闭。如果用 --myisam-recover 选项启动 MySQLd，MyISAM 表在打开时会自动检查，如果表没有恰当地关闭，就会修复该表。

（12）如果使用--update-state 选项运行 myisamchk，它标注表为已检查。myisamchk--fast 只检查那些没有这个标志的表。

（13）myisampack 可以打包 BLOB 和 VARCHAR 列。

（14）含有 VARCHAR 的表可以有固定或动态记录长度。

（15）VARCHAR 和 CHAR 列的总长度可以达 64KB。

2. MyISAM 选项配置

下面简单介绍几个对 MyISAM 表行为有影响的 MySQLd 选项。

（1）--myisam-recover=mode

设置 MyISAM 表崩溃时的自动恢复模式。

表 1-3 列举了 Mode 的取值。

表 1-3　Mode 的取值及含义

选　　项	描　　述
DEFAULT	与没有使用--myisam-recover 选项相同
BACKUP	如果在恢复过程中，数据文件被更改了，将 tbl_name.MYD 文件备份为 tbl_name-datetime.BAK
FORCE	即使.MYD 文件将丢失多行也进行强制恢复
QUICK	如果没有删除块，不要检查表中的行

（2）--delay-key-write=ALL

不刷新 MyISAM 表的键缓冲区。

如果指定这个选项，则不应该访问被另一个程序占用的 MyISAM 表（例如 MySQL 服务器或 myisamchk），这样做会破坏表的索引。

下列系统变量影响 MyISAM 表的行为：

- bulk_insert_buffer_size

用在块插入优化中的树缓冲区的大小，这是对每个线程的限制。

- myisam_max_sort_file_size

MySQL 在重建 MyISAM 表索引时能够使用的临时文件的最大长度，以字节为单位。如果文件大小超过这个值，建立索引的效率就会降低。

- myisam_sort_buffer_size

设置恢复表时使用的缓冲区的大小。

如果用--myisam-recover 自动恢复选项启动 MySQLd，那么当服务器打开一个 MyISAM 表时，如果发现以下任意一种情况：

- 表是否标注为崩溃。
- 表的打开计数器变量不为 0，并且服务器以--skip-external-locking 模式运行。

将会进行如下操作：

- 检查表是否有错。
- 如果服务器发现一个错误，它试着快速表修复（排序且不重新创建数据文件）。
- 如果修复因为数据文件中的一个错误而失败（例如，一个主键重复错误），服务器会再次尝试修复，并重建数据文件。

如果修复仍然失败，服务器用旧修复选项方法再重试一次修复（一行接一行地写，不排序）。这个方法应该能修复任何类型的错误，并且只需要很少的磁盘空间。

如果恢复不能够从之前完成的语句中恢复所有行，而且没有为--myisam-recover 选项值指定 FORCE，自动修复将会终止，并在错误日志中写一条错误信息：

```
Error: Couldn't repair table: test.g00pages
```

如果指定 FORCE，则会出现以下信息：

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

如果自动恢复模式使用 BACKUP，恢复进程会创建名为 tbl_name-datetime.BAK 的备份文件。这时可以使用一个脚本，自动把这些文件从数据库目录移到备份媒质上。

1.4.4　InnoDB 存储引擎

1. InnoDB 概述

InnoDB 给 MySQL 提供了具有提交、回滚和崩溃恢复能力的事务安全（ACID 兼容）存储引擎。InnoDB 锁定在行级并且也在 SELECT 语句提供一个 Oracle 风格的非锁定读。这些特色提高了多用户并发的性能。InnoDB 中行级锁定适合非常小的空间，因此没有在 InnoDB 中扩大锁定的需要，InnoDB 也支持外键约束。在 SQL 查询中，可以自由地将 InnoDB 类型表与其他类型的 MySQL 表混合使用。

InnoDB 是为处理大量数据而设计的。与其他基于磁盘的关系数据库引擎相比，它具有较高的 CPU 效率。

InnoDB 存储引擎完全与 MySQL 服务器整合，InnoDB 存储引擎为在内存中建立自己的缓冲池来缓存数据和索引。InnoDB 在一个表空间中存储表和索引，表空间可以包含数个文件（或原始磁盘分区）。这与 MyISAM 表不同，在 MyISAM 表中每个表被存在单独的文件中，因此即使在文件大小被限制在 2GB 的文件系统上，InnoDB 表也可以是任意大小的。

InnoDB 默认地被包含在 MySQL 二进制版本中。在 Windows Essentials installer 安装中，InnoDB 是 Windows 上 MySQL 的默认引擎。

InnoDB 被用来在众多需要高性能的大型数据库站点上产生。著名的 Internet 新闻站点 Slashdot.org 运行在 InnoDB 上。Mytrix，Inc.在 InnoDB 上存储超过 1TB 的数据，还有一些其他站点在 InnoDB 上处理平均每秒 800 次插入/更新的负荷。

2. InnoDB 配置

InnoDB 存储引擎默认是可以使用的。如果不想用 InnoDB 表，可以在 MySQL 配置文件 my.ini 中添加 skip-innodb 选项。

InnoDB 存储引擎管理两个重要的文件，即 InnoDB 表空间数据文件和它的日志文件。

如果没有对 InnoDB 进行配置，MySQL 会在 MySQL 数据目录下创建一个名为 ibdata1 的 10MB 大小的自动扩展数据文件，以及两个名为 ib_logfile0 和 ib_logfile1 的 5MB 大小的日志文件。

要建立 InnoDB 表空间文件，在 my.ini 配置文件的[MySQLd]节中使用 innodb_data_file_path 选项。innodb_data_file_path 的值应该为一个或多个数据文件规格的列表。如

果命名一个以上的数据文件，用分号（;）分隔它们，文件大小用 M 或者 G 后缀来指定说明单位是 MB 或者 GB。一个指定的数据文件规格可以包含文件名、文件大小、自动扩展选项和文件最大值，如：

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

autoextend 及其后面的属性只可用来配置 innodb_data_file_path 行中最后一个数据文件。

如果对最后的数据文件指定 autoextend 选项，当数据文件耗尽了表空间中的自由空间时，InnoDB 就会扩展此数据文件。扩展的幅度是每次 8MB。

配置文件中采用如下形式：

```
innodb_data_file_path=datafile_spec1[;datafile_spec2]...
```

例如：

```
[MySQLd]
innodb_data_file_path=ibdata1:10M:autoextend
```

这个设置配置一个可扩展大小的尺寸为 10MB 的单独文件，名为 ibdata1。没有给出文件的位置，所以默认的是在 MySQL 的数据目录内。

```
[MySQLd]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

这个配置指定的表空间在数据目录中包含一个名为 ibdata1 的固定尺寸 50MB 的数据文件和一个名为 ibdata2、大小为 50MB 的自动扩展文件。

InnoDB 并不感知最大文件尺寸，所以要注意文件系统，单个数据文件的大小不能超过文件系统支持的最大值。要为一个自动扩展数据文件指定最大尺寸，可以使用 max 属性。下列配置允许 ibdata1 涨到极限的 500MB：

```
[MySQLd]
innodb_data_file_path=ibdata1:10M:autoextend:max:500M
```

InnoDB 默认地在 MySQL 数据目录中创建表空间文件。要明确指定一个位置，可使用 innodb_data_home_dir 选项。例如，要使用两个名为 ibdata1 和 ibdata2 的文件，但要把它们创建到/ibdata，像如下一样配置 InnoDB：

```
[MySQLd]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

3. 创建 InnoDB 表

要创建一个 InnoDB 表，必须在表创建 SQL 语句中指定 ENGINE = InnoDB 或者 TYPE = InnoDB 选项：

```
CREATE TABLE customer (a INT, b CHAR (20), INDEX (a)) ENGINE=InnoDB;
CREATE TABLE customer (a INT, b CHAR (20), INDEX (a)) TYPE=InnoDB;
```

以上 SQL 语句在表空间的列上创建一个表和索引，表空间包含在 my.ini 中指定的数据

文件。此外，MySQL 在 MySQL 数据库目录下的 test 目录中创建一个名为 customer.frm 的文件。同时，InnoDB 会在自己的数据目录中为'test/customer'表添加一个表项。也就是说，可以在其他数据库创建一个具有相同名字 customer 的表，表的名字不会与 InnoDB 内的冲突。

可以对任何 InnoDB 表，通过使用 SHOW TABLE STATUS 语句，查询在 InnoDB 表空间内空闲空间的数量。表空间内空闲空间的数量出现在 SHOW TABLE STATUS 的输出结果内的 Comment 节中。例如：

```
SHOW TABLE STATUS FROM test LIKE 'customer'
```

注意，SHOW 只给出关于 InnoDB 表的大致统计情况。

默认情况下，每个连接到 MySQL 服务器的客户端最初是使用自动提交模式的，这个模式自动提交需要运行的每个 SQL 语句。要使用多语句事务，可以用 SQL 语句 SET AUTOCOMMIT = 0 禁止自动提交，并且用 COMMIT 和 ROLLBACK 来提交或回滚事务。如果 AUTOCOMMIT 保持打开状态，则可以在 START TRANSACTION 与 COMMIT 或 ROLLBACK 之间封装一个事务，下列的例子演示两个事务，第一个是被提交的，第二个是被回滚的。

```
shell> MySQL test
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 3.23.50-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
MySQL> CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A))
    -> ENGINE=InnoDB;
Query OK, 0 rows affected (0.00 sec)
MySQL> BEGIN;
Query OK, 0 rows affected (0.00 sec)
MySQL> INSERT INTO CUSTOMER VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
MySQL> COMMIT;
Query OK, 0 rows affected (0.00 sec)
MySQL> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.00 sec)
MySQL> INSERT INTO CUSTOMER VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
MySQL> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
MySQL> SELECT * FROM CUSTOMER;
+------+--------+
| A    | B      |
+------+--------+
|   10 | Heikki |
+------+--------+
1 row in set (0.00 sec)
```

在类似 PHP、Perl DBI/DBD、JDBC、ODBC 或者 MySQL 的标准 C 调用接口这样的 API 上，能够以字符串形式发送事务控制语句，如发送 COMMIT 到 MySQL 服务器，就像执行其他任何的 SQL 语句那样，诸如 SELECT 或 INSERT。一些 API 也提供单独的专门的事务提交和回滚函数或者方法。

4. 转换 MyISAM 表到 InnoDB

MyISAM 表可以根据需要转换为 InnoDB 表，但是不应该在 MySQL 数据库中把 MySQL 系统表转换为 InnoDB 类型。系统表总是 MyISAM 类型的。

如果需要所有（非系统）表都定义成 InnoDB 表，可以简单地把 default-table-type=innodb 行添加到 my.ini 文件的[MySQLd]段中，将 InnoDB 指定为默认的存储引擎。

InnoDB 不像 MyISAM 存储引擎对索引创建进行专门的优化。因此在导出表后创建索引并不能提高数据库的性能。将一个表转化为 InnoDB 表的最快捷的方法是使用 ALTER TABLE...ENGINE=INNODB 语句，将表指定为 InnoDB 存储引擎。也可以先新建一个结构和 MyISAM 表相同的 InnoDB 表，然后使用 INSERT INTO ... SELECT * FROM ...将 MyISAM 中的数据插入 InnoDB 表中。

在插入数据的过程中，如果在某个键上有 UNIQUE 约束，可以在导入阶段设置：SET UNIQUE_CHECKS=0，临时关掉唯一性检查以加速表的导入。对于大表，这节省了大量的磁盘 I/O 操作。

例如：

```
SET UNIQUE_CHECKS=0;
... import operation...
SET UNIQUE_CHECKS=1;
```

插入大量数据最好使用分段插入的方式，例如：

```
INSERT INTO newtable SELECT * FROM oldtable
   WHERE yourkey > something AND yourkey <= somethingelse;
```

所有记录已经被插入之后，可以重命名表。

在大表的转换中，虽然可以增加 InnoDB 缓冲池的大小来减少磁盘 I/O，提高数据插入效率，但是不要使用超过 80%的物理内存。

导入过程中，需要确保磁盘有足够的空间：InnoDB 表比 MyISAM 表需要更多的磁盘空间。如果一个 ALTER TABLE 耗尽了空间，它就开始一个回滚，回滚可能要几个小时。由于对插入操作做了特别的优化，因此，同样数据量的回滚要比插入多花 30 倍的时间。

在回滚失控的情况下，如果数据库中没有有价值的数据，比较明智的是杀掉数据库进程而不是等几百万个磁盘 I/O 完成。

1.5 小　　结

通过本章的学习，读者应该对数据库建立了基本的概念。同时，应该去试着了解如 SQL Server、Oracle 等常用数据库，重点理解 MySQL 数据库，明白与其他数据库相比，MySQL 数据库的优势所在。同时，需要对 MySQL 数据库的体系结构有一定了解，理解 MySQL 的各种存储引擎及其特点，重点掌握 MyISAM 存储引擎和 InnoDB 存储引擎。

第 2 章　MySQL 基本操作

本章主要介绍 MySQL 的基础知识，包括下载、安装 MySQL 的方法。同时，为了方便 MySQL 的使用，还推荐了 MySQL GUI 工具。2.3 节比较详细地介绍 MySQL 命令行客户端和 MySQL 服务端的使用方法。2.4 节介绍一些常用 MySQL 实用工具程序，这些程序的功能包括对 MySQL 的管理、备份、检查等。

2.1　获得 MySQL

可以在 MySQL 的官方网站获得 MySQL 的安装程序。其官方网站为 www.mysql.com。MySQL 5.1 的下载地址为 http://dev.mysql.com/downloads/mysql/5.1.html。在 MySQL 5.1 的众多版本中，可以选择适合自己的版本进行安装。

在下载网页中，列出了许多版本的安装包，如图 2-1 所示为网页的部分截图。

Windows downloads (platform notes)

Windows Essentials (x86)	5.1.20-beta	25.0M	Download | Pick a mirror MD5: c688a07de13a5ale2e7706e34c582906
Windows ZIP/Setup.EXE (x86)	5.1.20-beta	52.7M	Download | Pick a mirror MD5: c42980e4253663a417085ca883fd1f97 | Signature
Without installer (unzip in C:\)	5.1.20-beta	55.8M	Download | Pick a mirror MD5: 352539a7daee6a560a79b3b416547f8f | Signature

Windows x64 downloads (platform notes)

Windows Essentials (AMD64 / Intel EM64T)	5.1.20-beta	30.4M	Download | Pick a mirror MD5: e363e3cbf8a03a186b9ad26bd96cb0d8
Windows ZIP/Setup.EXE (AMD64 / Intel EM64T)	5.1.20-beta	55.5M	Download | Pick a mirror MD5: 551cb409e70449eef86eb8688eb43b72 | Signature
Without installer (AMD64 / Intel EM64T)	5.1.20-beta	68.8M	Download | Pick a mirror MD5: d2490119f1aa81b4fd54a862cc5cc437 | Signature

Linux (non RPM packages) downloads (platform notes)

Linux (x86)	5.1.20-beta	75.8M	Download | Pick a mirror MD5: 13f38a8dfd820d6fa391ffd9c737cd6f | Signature
Linux (AMD64 / Intel EM64T)	5.1.20-beta	80.0M	Download | Pick a mirror MD5: 15def3c761140bfddafd342df3ba02b9 | Signature
Linux (POWER / PowerPC, 32-bit)	5.1.20-beta	92.5M	Download | Pick a mirror MD5: 5bb5dd4f1f4cedf703433f7642dd0c6b | Signature
Linux (S/390X)	5.1.20-beta	81.0M	Download | Pick a mirror MD5: ae77595418365cb86cac001943e4a735 | Signature

Linux (non RPM, Intel C/C++ compiled, glibc-2.3) downloads

MySQL is pleased to make available offerings of the MySQL Community Server compiled with the Intel CC compiler. Internal tests show that editions of the MySQL Community Server compiled with the Intel CC compiler exhibit faster performance on Intel hardware than those compiled with the standard gcc compiler. Feedback is welcome.

For older MySQL packages compiled with Intel C/C++, you might need some supplementary downloads.

图 2-1　部分下载安装包

在 Windows 安装中，分为 32 位和 64 位两种。可以根据 CPU 的情况进行选择。

Windows Essentials 是 MySQL 的基本安装，仅包含在 Windows 中安装 MySQL 的最少文件，包括安装向导，但不包括可选组件，如嵌入式数据库和基准组件。

Windows Zip/Setup.exe 是指 MySQL 的完全安装，包括所有可选组件。

Without installer 含有 Windows Zip/Setup.exe 的所有组件，但是不含有安装向导，所有

的安装和配置需要手工进行。

在 Linux 版本中，也可以选择源代码安装和 RPM 安装。可以根据不同的 CPU 类型，下载适合自己的 MySQL 版本。

2.2　MySQL 的安装和配置

MySQL 安装向导和 MySQL 配置向导可以帮助新用户快速完成 MySQL 的安装和配置过程。在 MySQL 的完全安装包中，MySQL 安装向导和 MySQL 配置向导都是可用的，同时这两个向导在大部分标准的 MySQL 安装程序中被推荐使用。

2.2.1　Windows 下二进制包安装

MySQL 可以在 Windows 95/98/Me/2000/NT/XP 和 Windows 2003 下运行。如果需要将 MySQL 作为服务器运行，强烈建议使用基于 Windows NT 的操作系统，如 Windows 2000。

在安装包下载完成后，即可进行安装。这里以 MySQL 5.1 的 Windows Zip/Setup.exe 安装版本为例，对安装过程进行说明。图 2-2 显示了 MySQL 安装程序的欢迎界面。

图 2-3 中，Typical 为典型安装，仅安装一些公共组件，包括 MySQL 服务器，mysql 命令行客户端和命令行实用程序，命令行客户端和实用程序包括 mysqldump 和 myisamchk；Complete 为完全安装，包括的组件包括嵌入式服务器库、基准套件、支持脚本和文档；Custom 是自定义安装，可以有选择地安装自己需要的组件，如果要选择 MySQL 的安装路径，必须选中该单选按钮。这里可以选中 Custom 单选按钮，单击 Next 按钮。

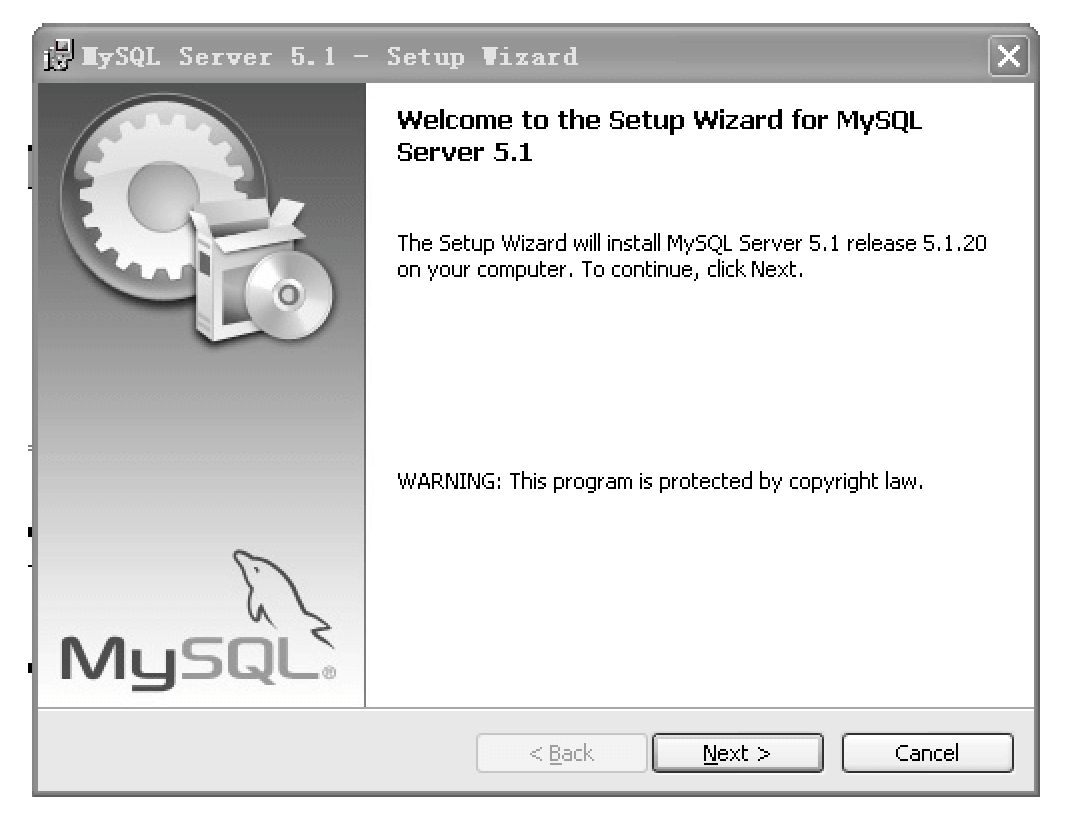




图 2-2　MySQL 安装向导

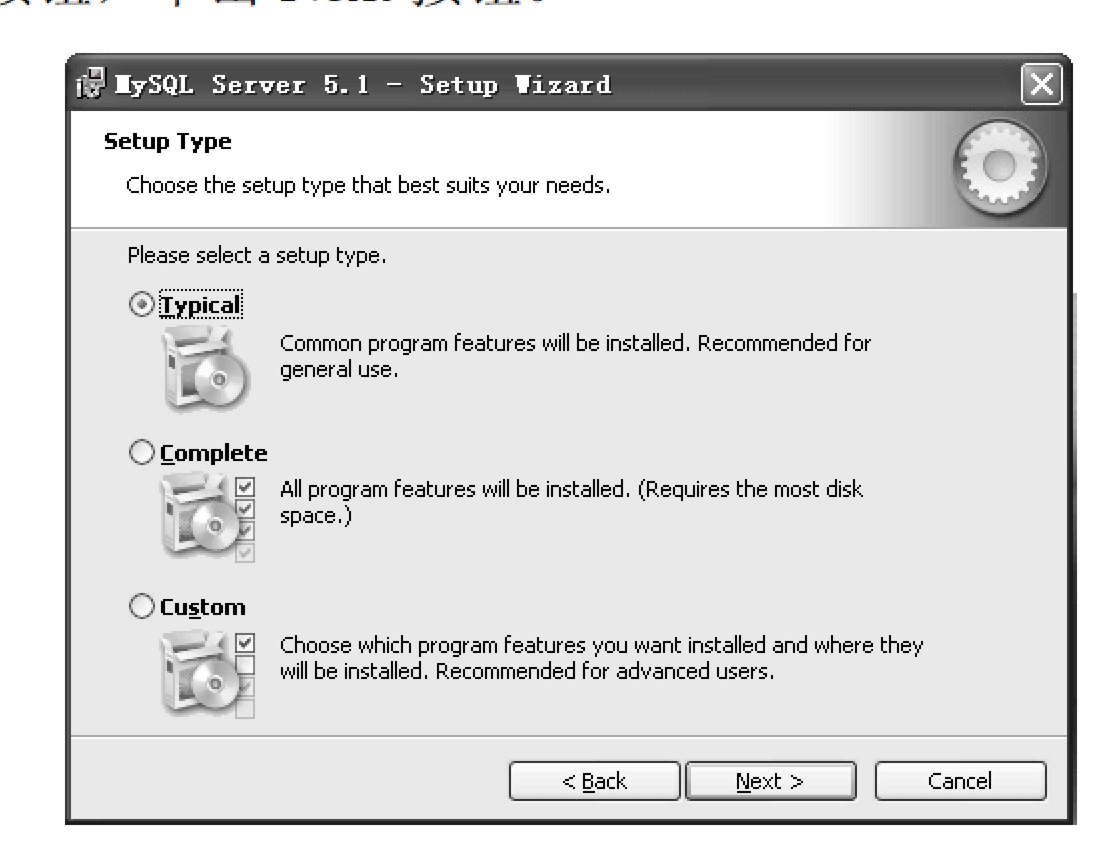


图 2-3　MySQL 安装类型选择

如图 2-4 所示，单击 Change 按钮可以改变安装路径。同时在列表框中选择需要安装的组件。

单击 Next 按钮，然后选择 Install。安装完成后，出现如图 2-5 所示的对话框。

如图 2-5 所示，选中 Configure the MySQL Server now 复选框，单击 Finish 按钮，进行服务器的配置。配置向导会将服务器的配置存放到 my.ini 文件中，避免了手工输入的麻烦。

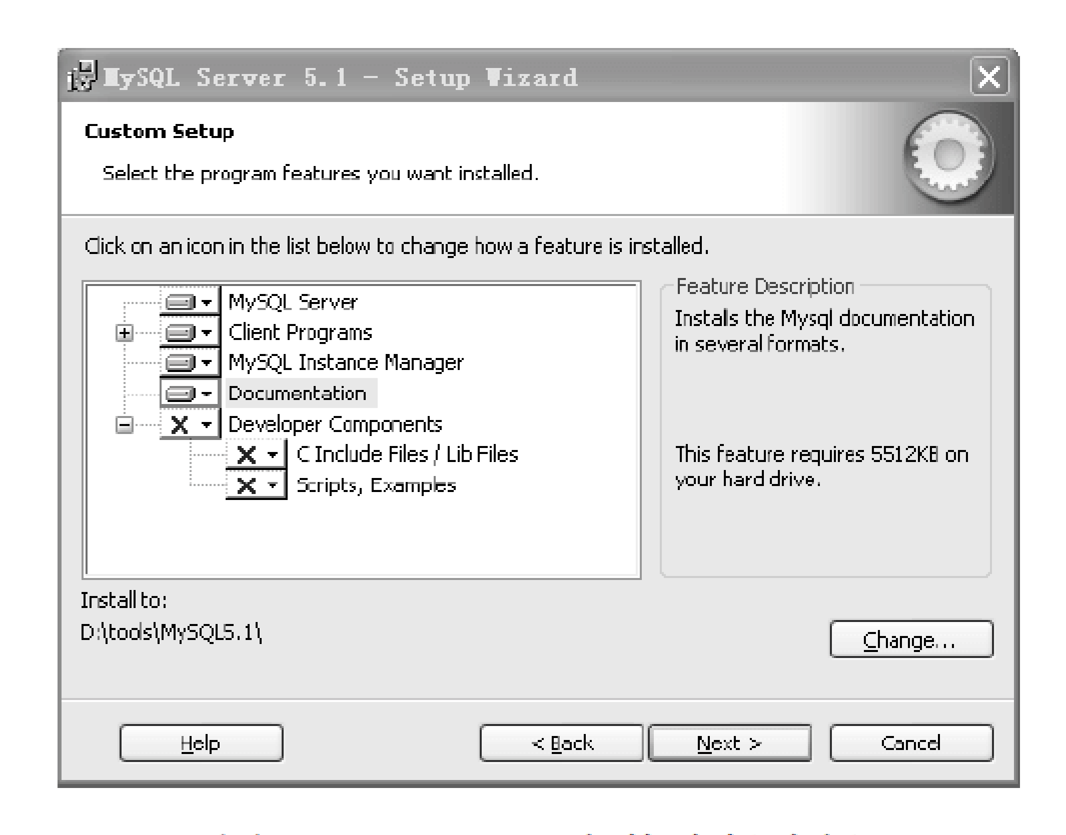



图 2-4　MySQL 安装路径选择

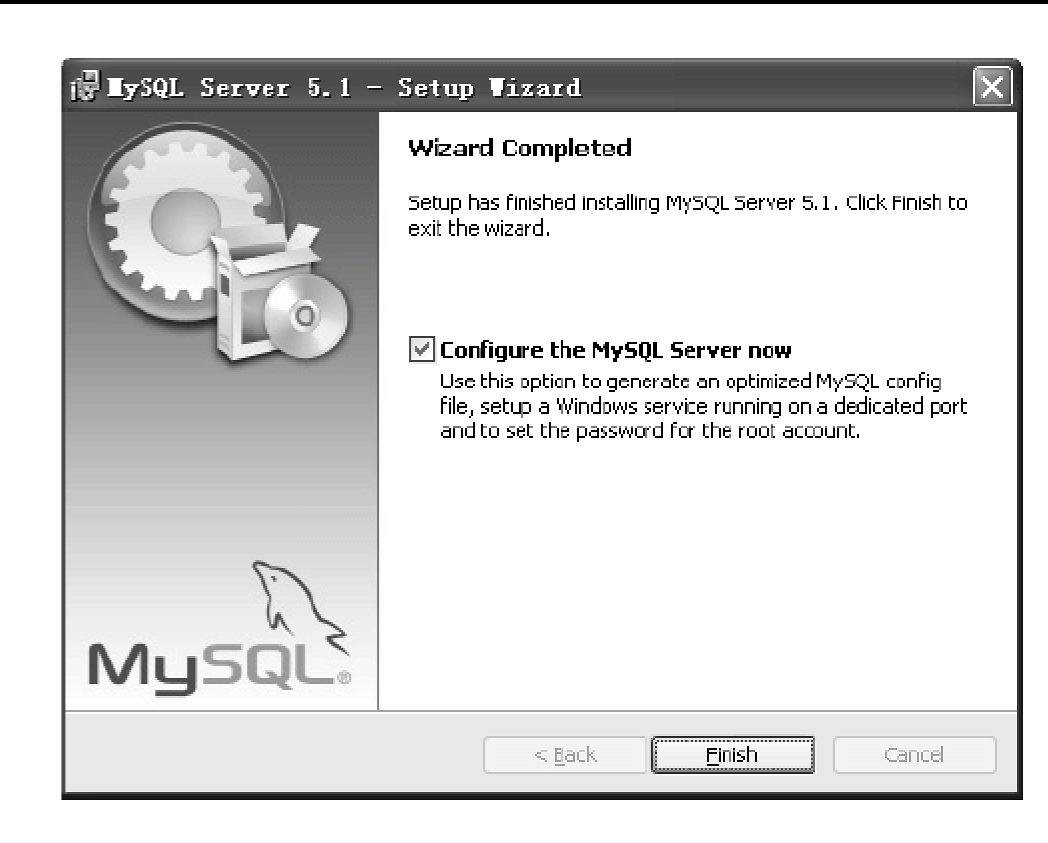


图 2-5　“配置服务器”对话框

初始安装后，服务器配置分为两种，Detailed Configuration（详细配置）和 Standard Configuration（标准配置），如图 2-6 所示。Standard Configuration（标准配置）选项适合想要快速启动 MySQL 而不必考虑服务器配置的新用户。详细配置选项适合想要更加细粒度控制服务器配置的高级用户。这里为了更加清晰地说明 MySQL 的配置选项，选中 Detailed Configuration（详细配置）单选按钮。在实际安装过程中，如果 MySQL 服务器仅作为开发使用，则选中 Standard Configuration（标准配置）单选按钮更为方便。选中 Detailed Configuration（详细配置）单选按钮，单击 Next 按钮。

如图 2-7 所示，可以选择 3 种服务器类型，选择哪种服务器将影响到 MySQL Configuration Wizard（配置向导）对内存、硬盘和处理器的使用策略。

- ❑ Developer Machine（开发机器）：该选项代表典型个人用桌面工作站。假定机器上运行着多个桌面应用程序。将 MySQL 服务器配置成使用最少的系统资源。
- ❑ Server Machine（服务器）：该选项代表服务器，MySQL 服务器可以同其他应用程序一起运行，例如 FTP、E-mail 和 Web 服务器。MySQL 服务器配置成使用适当比例的系统资源。
- ❑ Dedicated MySQL Server Machine（专用 MySQL 服务器）：该选项代表只运行 MySQL 服务的服务器。假定没有运行其他应用程序，MySQL 服务器配置成使用所有可用系统资源。

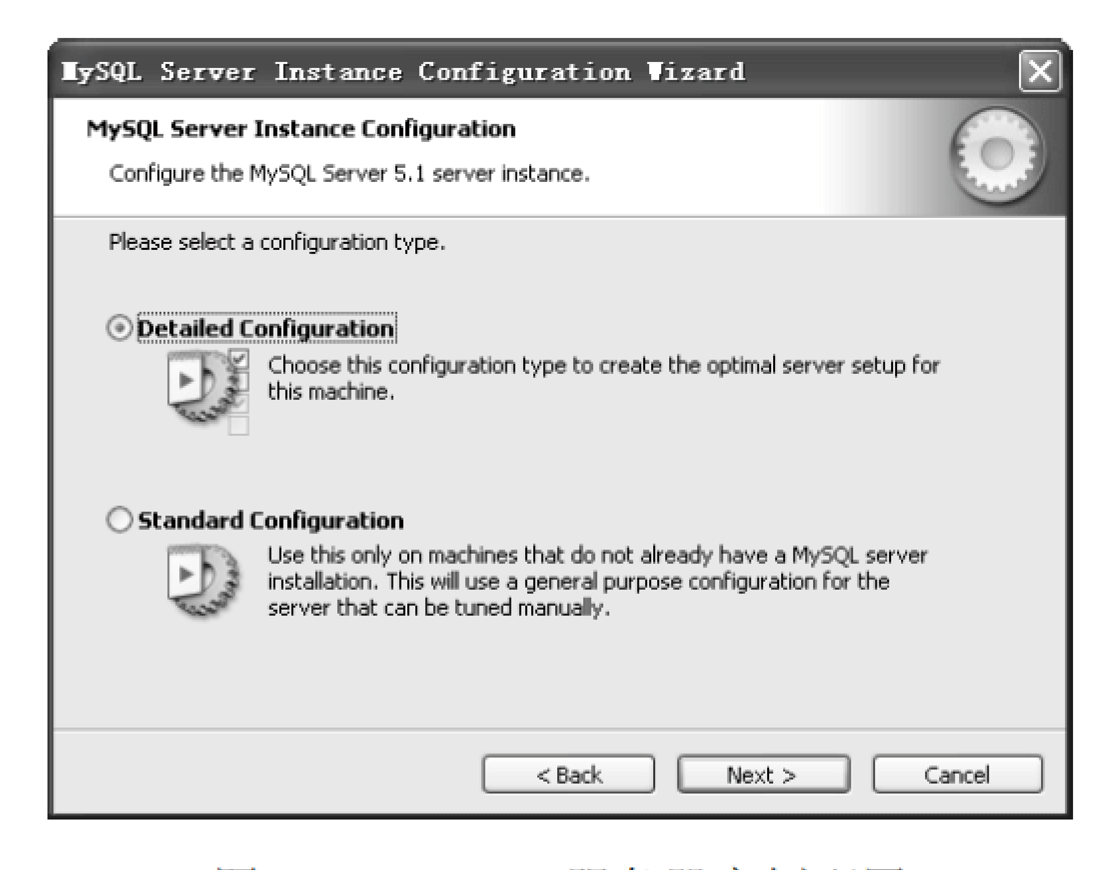


图 2-6　MySQL 服务器实例配置



图 2-7　MySQL 服务器类型选择

这里以 Developer Machine（开发机器）为例，选中 Developer Machine 单选按钮，单击 Next 按钮，出现 Database Usage（数据库用途）对话框，如图 2-8 所示。选择数据库的用途，这里可以指出创建 MySQL 表时使用的存储引擎的类型。通过该选项，可以选择是否使用 InnoDB 存储引擎，以及 InnoDB 占用多大比例的服务器资源。

- ❑ Multifunctional Database（多功能数据库）：选中该单选按钮，则同时使用 InnoDB 和 MyISAM 存储引擎，并在两个引擎之间平均分配资源。建议经常使用两个存储引擎的用户选中该单选按钮。
- ❑ Transactional Database Only（只是事务处理数据库）：该单选按钮同时使用 InnoDB 和 MyISAM 存储引擎，但将大多数服务器资源指派给 InnoDB 存储引擎。建议主要使用 InnoDB 只偶尔使用 MyISAM 的用户选中该单选按钮。
- ❑ Non-Transactional Database Only（只是非事务处理数据库）：该单选按钮完全禁用 InnoDB 存储引擎，将所有服务器资源指派给 MyISAM 存储引擎。建议不使用 InnoDB 的用户选中该单选按钮。

这里选中 Multifunctional Database（多功能数据库）单选按钮，单击 Next 按钮。

在图 2-9 中，InnoBD Tablespace Settings（InnoDB 表空间配置）允许指定 InnoDB 表空间文件的物理路径。如果系统有较大的空间或较高性能的存储设备（例如 RAID 存储系统），则最好将表空间文件单独放到一个位置。

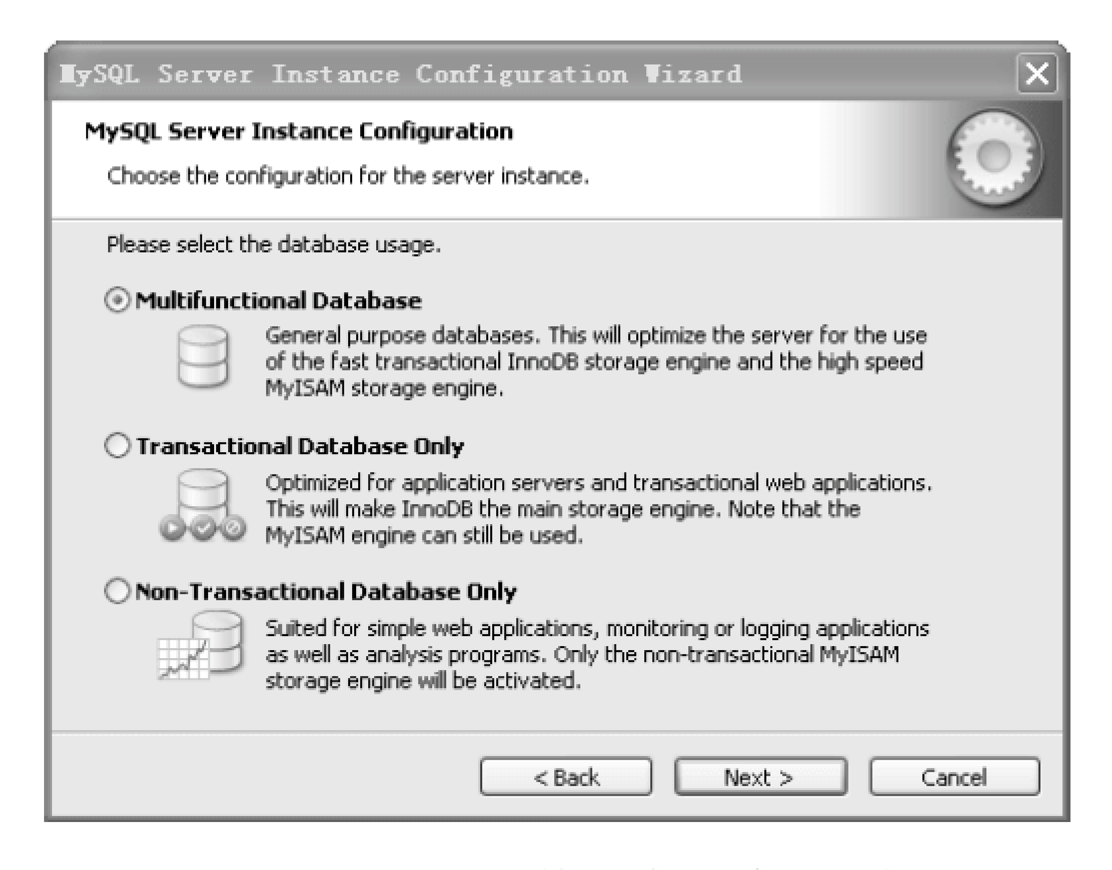


图 2-8　MySQL 数据库用途对话框

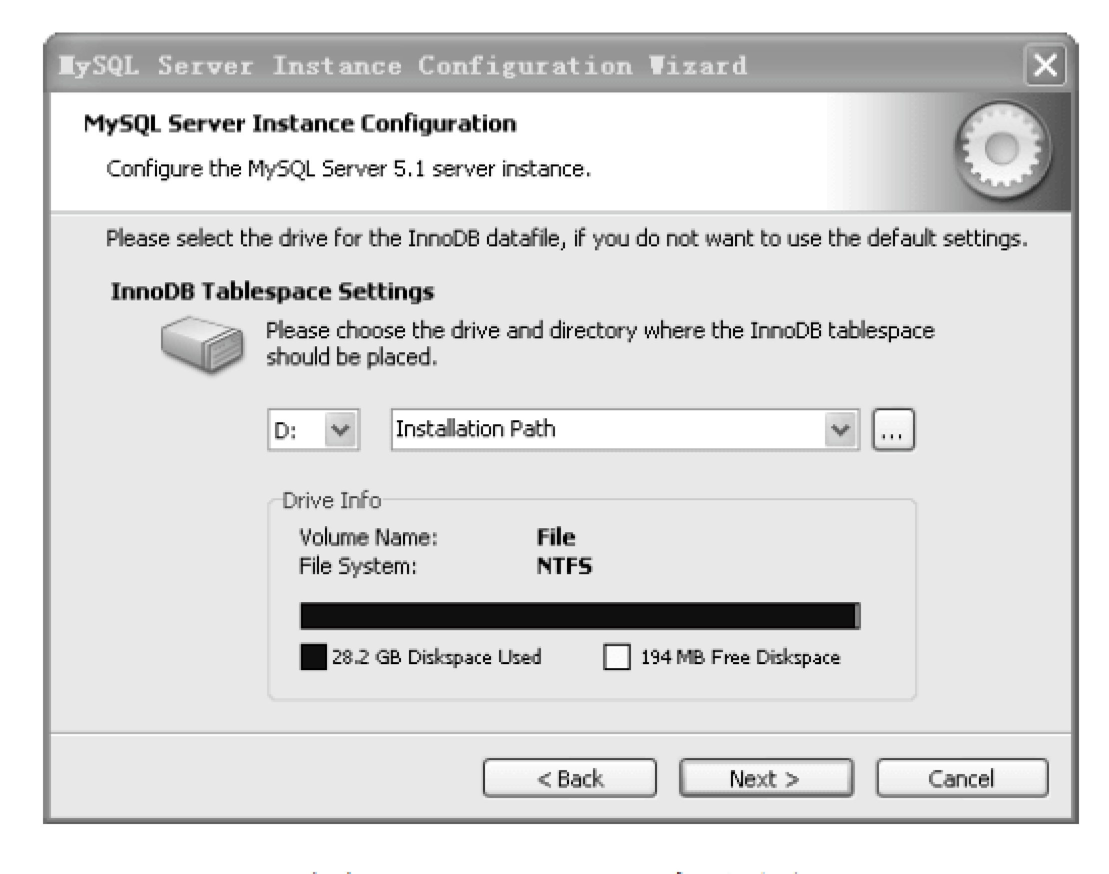


图 2-9　InnoDB 表配置

要想更改 InnoDB 表空间文件的默认位置，从驱动器下拉列表中选择一个新的驱动器，并从路径下拉列表中选择新的路径。要想创建路径，单击...按钮。

这里选用默认值，不做更改。单击 Next 按钮。进行 Concurrent Connections（并发连接）配置。可以选择服务器的使用方法，并根据情况限制并行连接的数量。还可以手动设置并行连接的限制。

MySQL 连接数配置如图 2-10 所示。其中包括 3 个单选按钮，分别介绍如下。

- ❑ Decision Support（决策支持）（DSS）/OLAP：如果服务器不需要大量的并行连接可以选中该单选按钮。假定最大连接数目设置为 100，则平均并行连接数为 20。
- ❑ Online Transaction Processing（联机事务处理）（OLTP）：如果服务器需要大量的并行连接则选中该单选按钮。最大连接数设置为 500。

- ❑ Manual Setting（人工设置）：选中该单选按钮可以手动设置服务器并行连接的最大数目。从前面的下拉列表框中选择并行连接的数目，如果期望的数目不在列表中，则在下拉列表框中输入最大连接数。

选择合适的选项后，单击 Next 按钮，进入 Networking Options（网络选项）对话框。在 Networking Options（网络选项）对话框中可以启用或禁用 TCP/IP 网络，并配置用来连接 MySQL 服务器的端口号。默认情况启用 TCP/IP 网络。要想禁用 TCP/IP 网络，取消选中 Enable TCP/IP Networking 复选框。默认使用 3306 端口。要想再访问 MySQL 使用的端口，从下拉列表框中选择一个新端口号或直接向下拉列表框输入新的端口号。如果选择的端口号已经被占用，将提示确认选择的端口号。单击 Next 按钮，进入 Character Set（字符集）配置对话框，如图 2-11 所示。

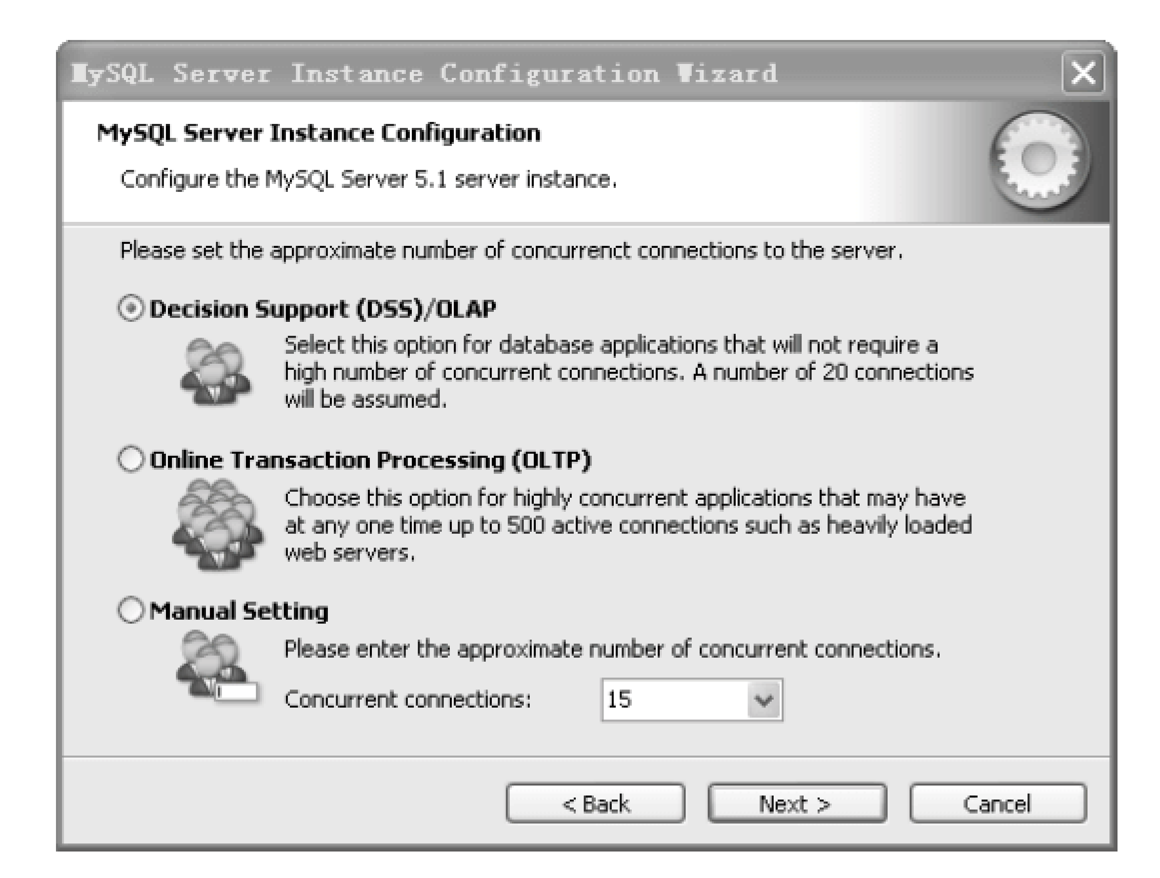


图 2-10　MySQL 连接数配置

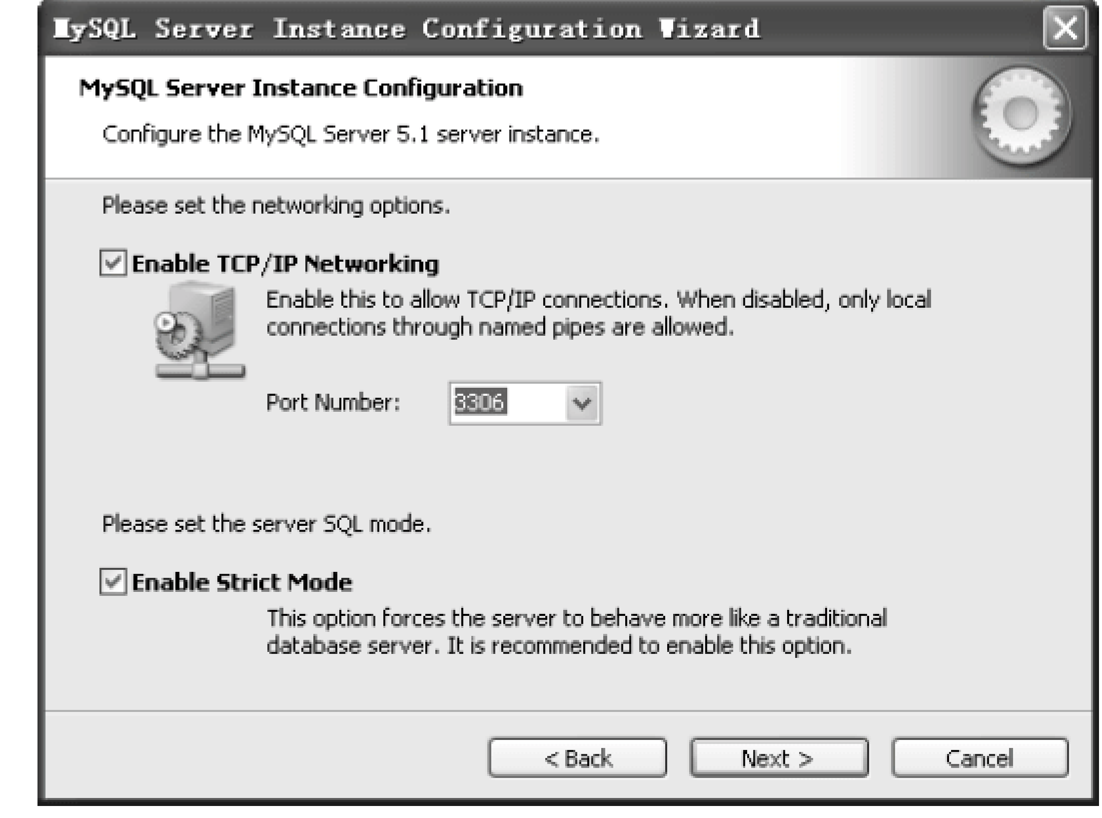


图 2-11　端口及模式配置

MySQL 服务器支持多种字符集，可以设置适用于所有表、列和数据库的默认服务器字符集。使用 Character Set（字符集）对话框来更改 MySQL 服务器的默认字符集，如图 2-12 所示。

- ❑ Standard Character Set（标准字符集）：如果想要使用 Latin1 作为默认服务器字符集，则选中该单选按钮。Latin1 用于英语和许多西欧语言。
- ❑ Best Support For Multilingualism（支持多种语言）：如果想要使用 UTF8 作为默认服务器字符集，则选中该单选按钮。UTF8 可以将不同语言的字符存储为单一的字符集。
- ❑ Manual Selected Default Character Set/Collation（人工选择的默认字符集/校对规则）：如果想要手动选择服务器的默认字符集，请选中该单选按钮。从下拉列表框中选择期望的字符集。

选择适合自己的选项，单击 Next 按钮，进入 Windows Options（Windows 选项）对话框。在这里，可以将 MySQL 设置为 Windows 服务，这里取服务名为 MySQL51，并且将 MySQL 服务配置为自启动服务。同时将 MySQL 工具集所在路径注册到 Windows 环境遍历 Path 中，如图 2-13 所示。

单击 Next 按钮，进入 Security Options（安全选项）对话框，进行 root 用户密码设置，

如图 2-14 所示。同时也可以创建匿名用户。

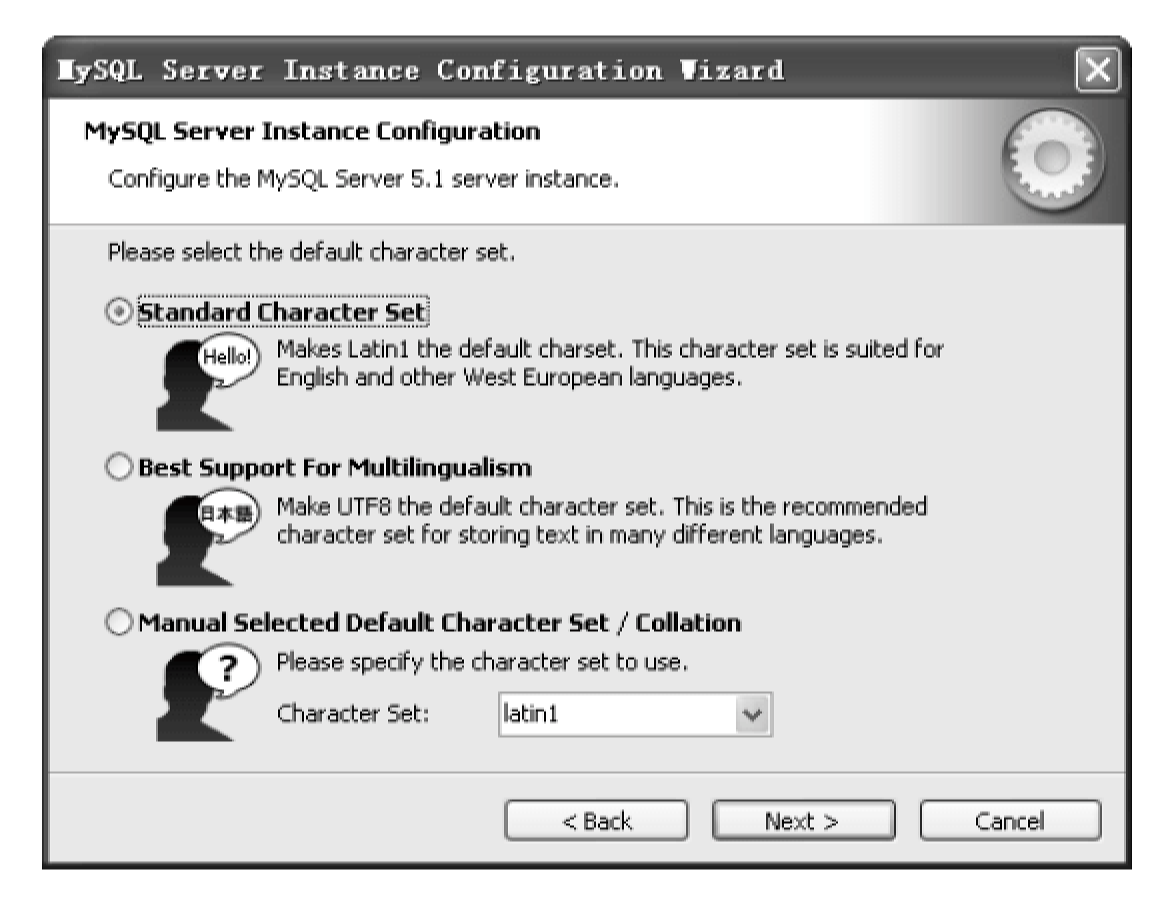


图 2-12　MySQL 字符集设置



图 2-13　服务安装及环境变量注册

设置完成后，单击 Next 按钮，出现 Execute（执行）对话框，单击 Execute 按钮，执行刚才配置的操作。执行完成后，单击 Finish 按钮，退出配置向导，如图 2-15 所示。



图 2-14　密码设置

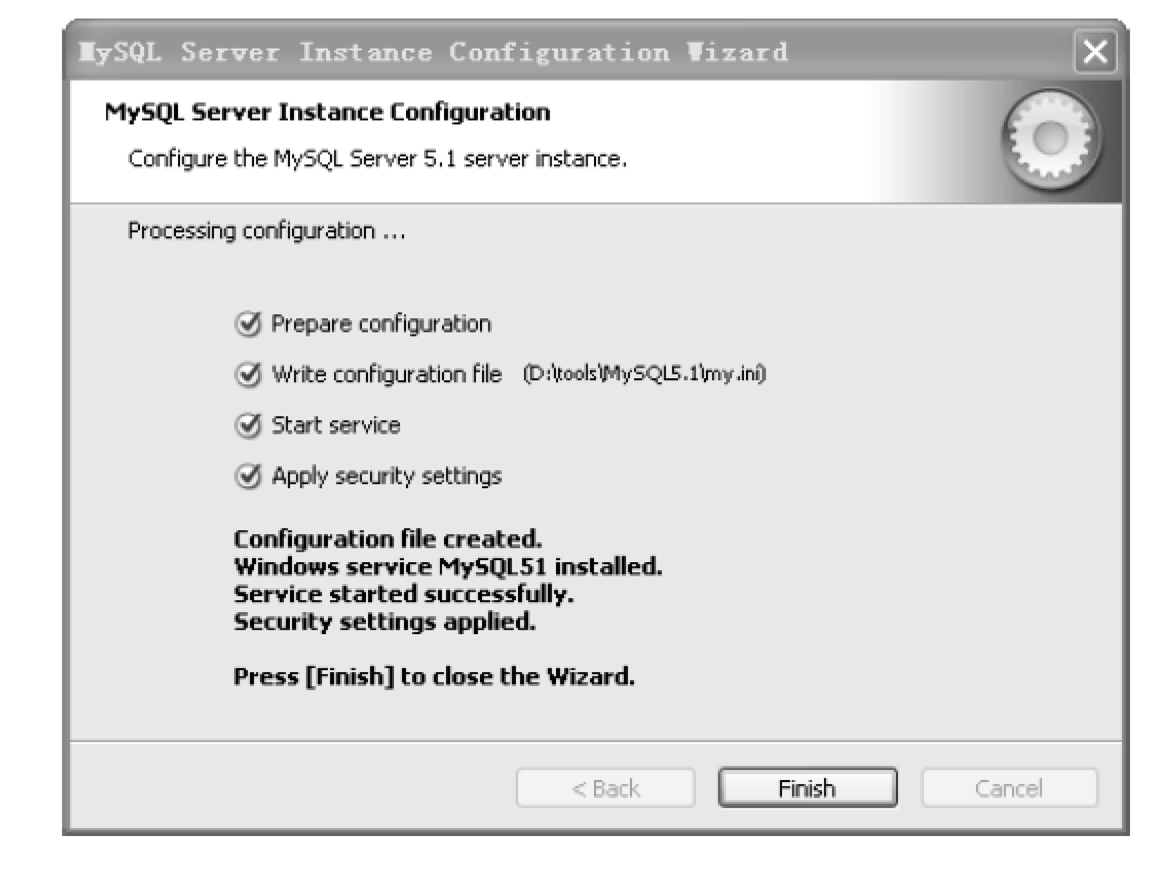


图 2-15　完成安装

到此为止，MySQL 服务器配置全部完成，MySQL Configuration Wizard（配置向导）将 my.ini 文件放到 MySQL 服务器的安装目录中。

2.2.2　重新配置 MySQL 服务器

在有些情况下，可能有需要更改 MySQL 服务器的配置选项。可以使用 MySQLInstanceConfig 命令启动配置向导。如果已经将 MySQL 安装目录下的 bin 文件夹注册到 Path 环境变量中，可以在命令行输入“MySQLInstanceConfig”，就会弹出 MySQL 配置向导。在 MySQL 配置向导中，允许对服务器进行重新配置，如图 2-16 所示。

选择 MySQL Server 5.1 的服务器，单击 Next 按钮，进入如图 2-17 所示的界面。

选中 Reconfigure Instance（重新配置）单选按钮可以对数据库实例进行新的配置。以后的选项与第一次配置服务器时相同。

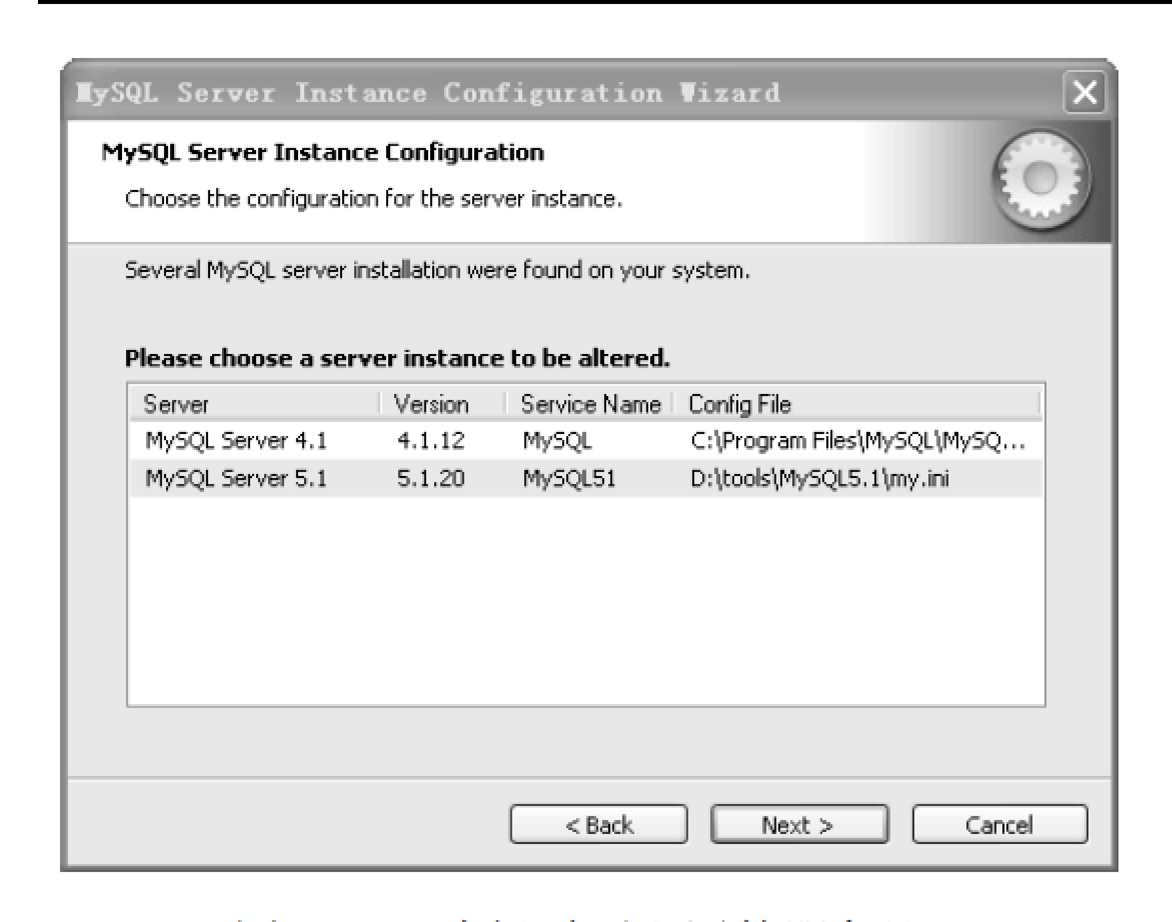



图 2-16　选择需要配置的服务器



图 2-17　重新配置服务器实例

2.3　MySQL GUI 的安装和使用

MySQL AB 公司为了提高 MySQL 服务器的易用性，为 MySQL 提供了一系列 GUI 工具，包括 MySQL GUI Tools 和 MySQL Workbench。这些工具可以很方便地对 MySQL 进行操作、管理、建模，并能大幅度提高工作效率。这些工具可以从 MySQL 的官方网站 www.mysql.com 下载。最新版本的 MySQL GUI Tools 下载地址为 http://dev.mysql.com/downloads/gui-tools/5.0.html。在 MySQL GUI Tools 工具中，包括 MySQL Administrator、MySQL Query Browser 和 MySQL Migration Toolkit 3 个程序。MySQL Administrator 用于管理 MySQL 数据库，MySQL Query Browser 用于管理查询数据，MySQL Migration Toolkit 可以方便其他数据库向 MySQL 数据库进行迁移。

MySQL Workbench 是一个基于 MySQL 的数据库建模工具，其下载地址为 http://ftp.ntu.edu.tw/pub/MySQL/Downloads/MySQLGUITools/。这些程序的安装都很简单，故不再介绍。下面主要介绍这些程序的使用方法。

2.3.1　MySQL Administrator

MySQL Administrator 提供了图形化的界面来管理和配置 MySQL 服务器。

MySQL GUI Tools 安装完成后，即可进入 MySQL Administrator。首先，需要填写连接的 MySQL 数据库，如图 2-18 所示。

单击 OK 按钮后，进入 MySQL Administrator 主窗口，如图 2-19 所示。

MySQL Administrator 的主要功能都列在左边的列表框中。

- ❑ Server Information 中可以看到当前服务器的运行状态等信息。
- ❑ Service Control 中可以开启、关闭 MySQL 服务，并且可以对服务进行简单的配置。
- ❑ Startup Variables 中可以设置许多服务器参数，包括配置 MyISAM、InnoDB 存储引擎、服务器性能配置、安全和网络设置等。

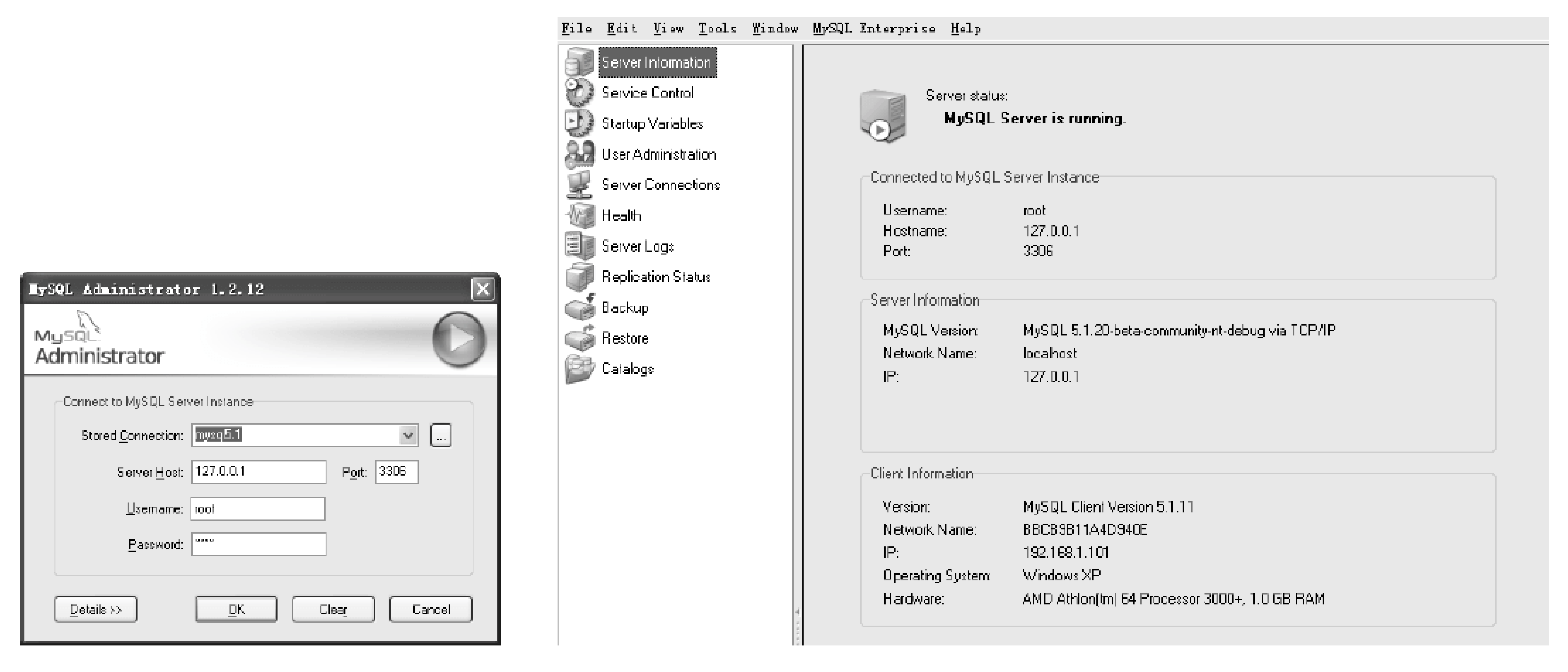


图 2-18　MySQL Administrator 的登录界面　　　　图 2-19　MySQL Administrator 主窗口

- ❑ User Administration 中可以增加、删除和管理用户。
- ❑ Server Connections 中可以查看当前有哪些用户连接到了服务器，以及相关连接信息。
- ❑ Health 中可以实时监视服务器状态，查看连接利用率，查询数量，缓冲利用等情况。
- ❑ Server Logs 可以非常方便地查看系统的各种日志。
- ❑ Replication Status 显示了当前服务器的复制状态。
- ❑ Backup 中可以新建、管理一个数据库备份项目。实施数据库的备份工作。
- ❑ Restore 可以使用 Backup 所得到的备份文件来还原数据库。
- ❑ Catalogs 中显示了当前 MySQL 服务器中所有的数据库。如图 2-20 所示，在 Catalogs 中还可以编辑、增加、删除表和进行数据库的维护，但不能查看表中的数据。如果需要查看数据，可使用 MySQL Query Browser 工具。

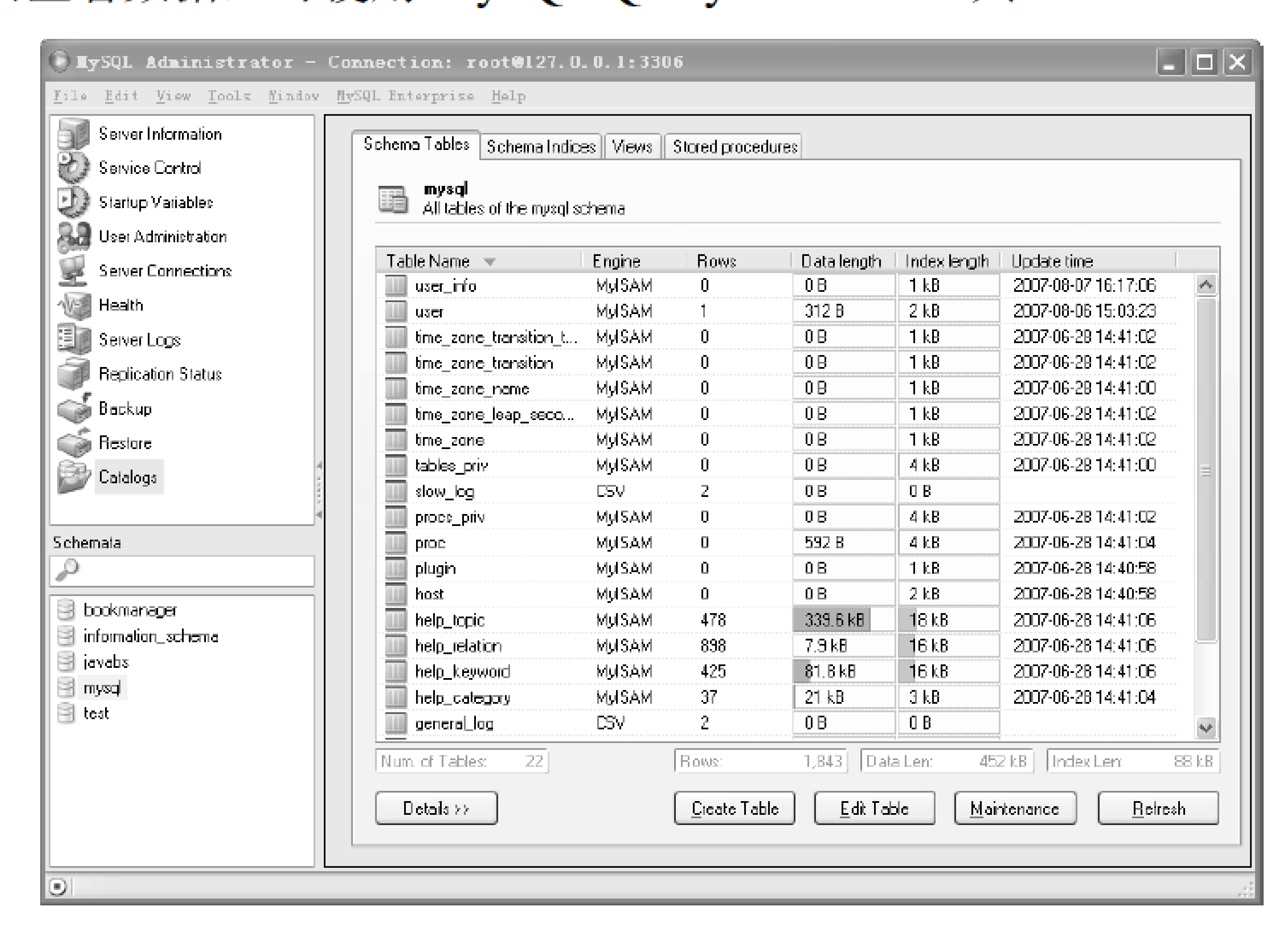


图 2-20　MySQL Administrator 数据库管理

2.3.2　MySQL Query Browser

MySQL Query Browser 提供了图形化的方式来访问数据库的数据。如图 2-21 所示为 MySQL Query Browser 的登录界面。

登录 MySQL Query Browser 后，其主界面如图 2-22 所示。

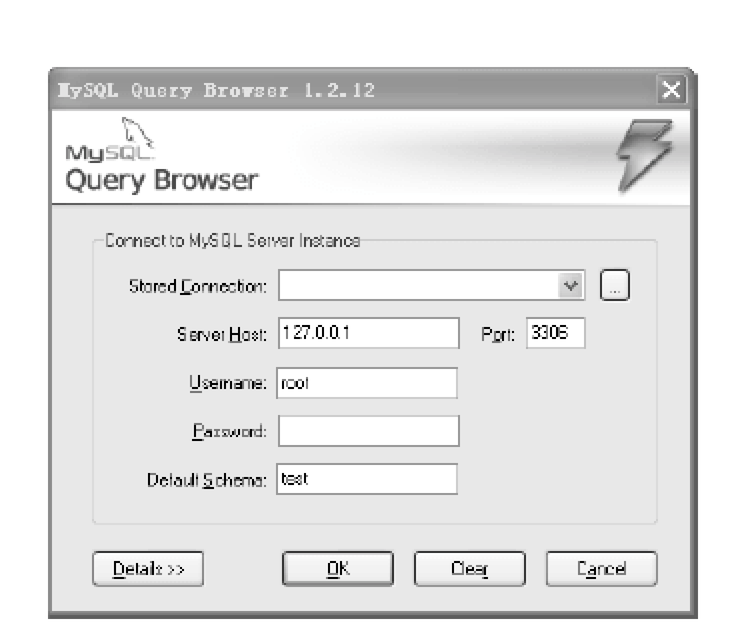


图 2-21　MySQL Query Browser 的登录界面

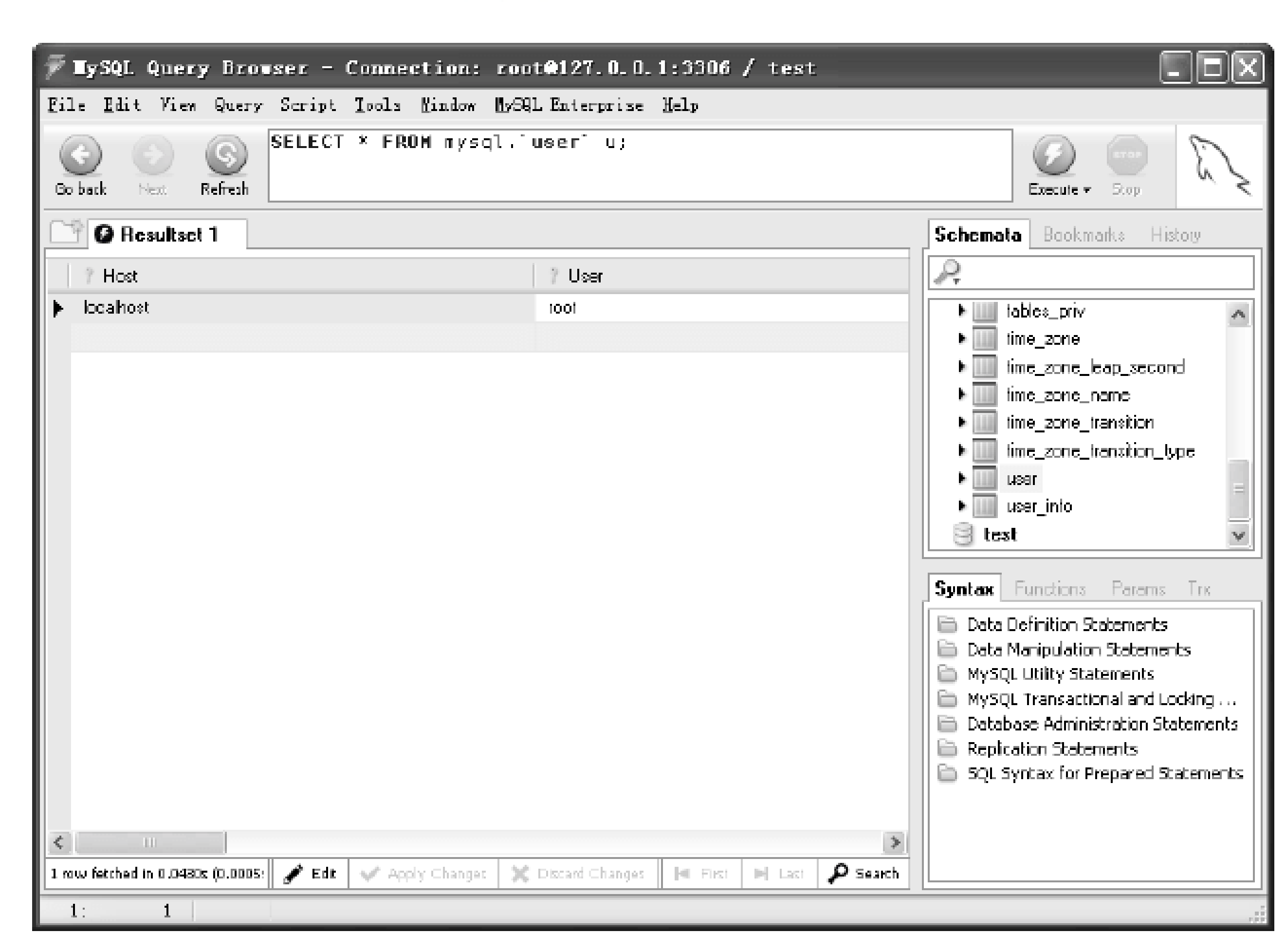


图 2-22　MySQL Query Browser 主界面

MySQL Query Browser 界面简洁明了，方便易用。右边的数据库列表中，可以选择需要访问的数据库和表。在中间的主窗口中，显示所选择表的数据，可以进行插入、编辑和删除等操作，也能够在上方的文本框中执行相应的 SQL 语句。

2.3.3　MySQL Migration Toolkit

MySQL Migration Toolkit 是一款数据迁移工具，它允许将其他数据库的数据迁移到 MySQL 中，方便用户从一个数据库平台向 MySQL 平台过渡。MySQL Migration Toolkit 提供了一个向导方式进行数据迁移操作。图 2-23 是 MySQL Migration Toolkit 的一个工作界面。限于篇幅有限，这里不作过多的介绍。

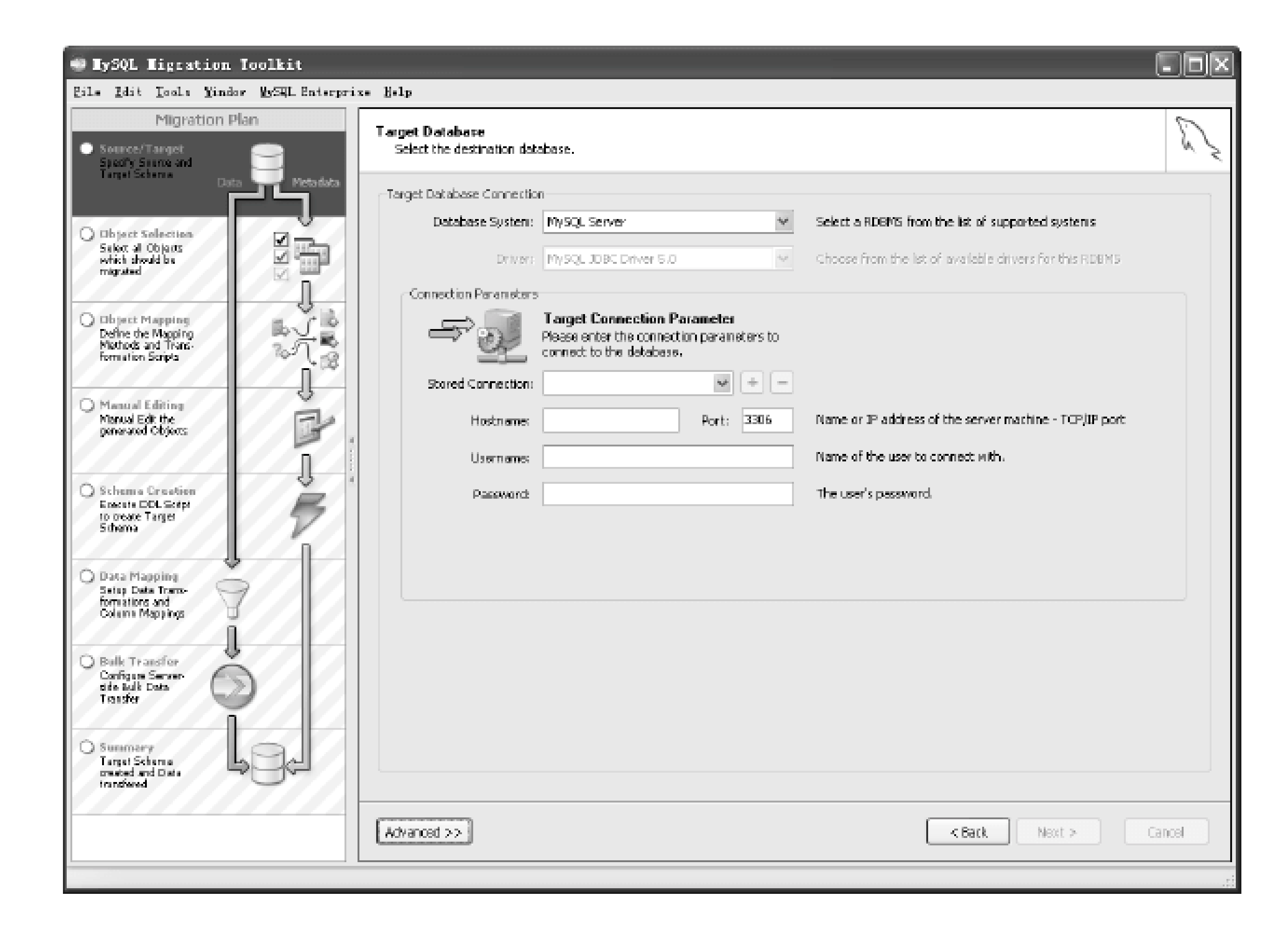


图 2-23　MySQL Migration Toolkit 工作界面

2.3.4　MySQL Workbench

MySQL Workbench 是一款数据库建模工具，可以和 MySQL 服务器很好地整合在一起。它提供了数据库设计功能，并能将数据库导出为 SQL，也能够实现 MySQL 数据库和 MySQL Workbench 设计的同步，支持 MySQL 数据的逆向工程。图 2-24 显示了 MySQL Workbench 的一个数据库设计。由于 MySQL Workbench 目前还不太成熟，仅提供了 Alpha 版本，因此，并不推荐大家使用。如果条件允许，建议使用 Power Designer，详见第 4 章。

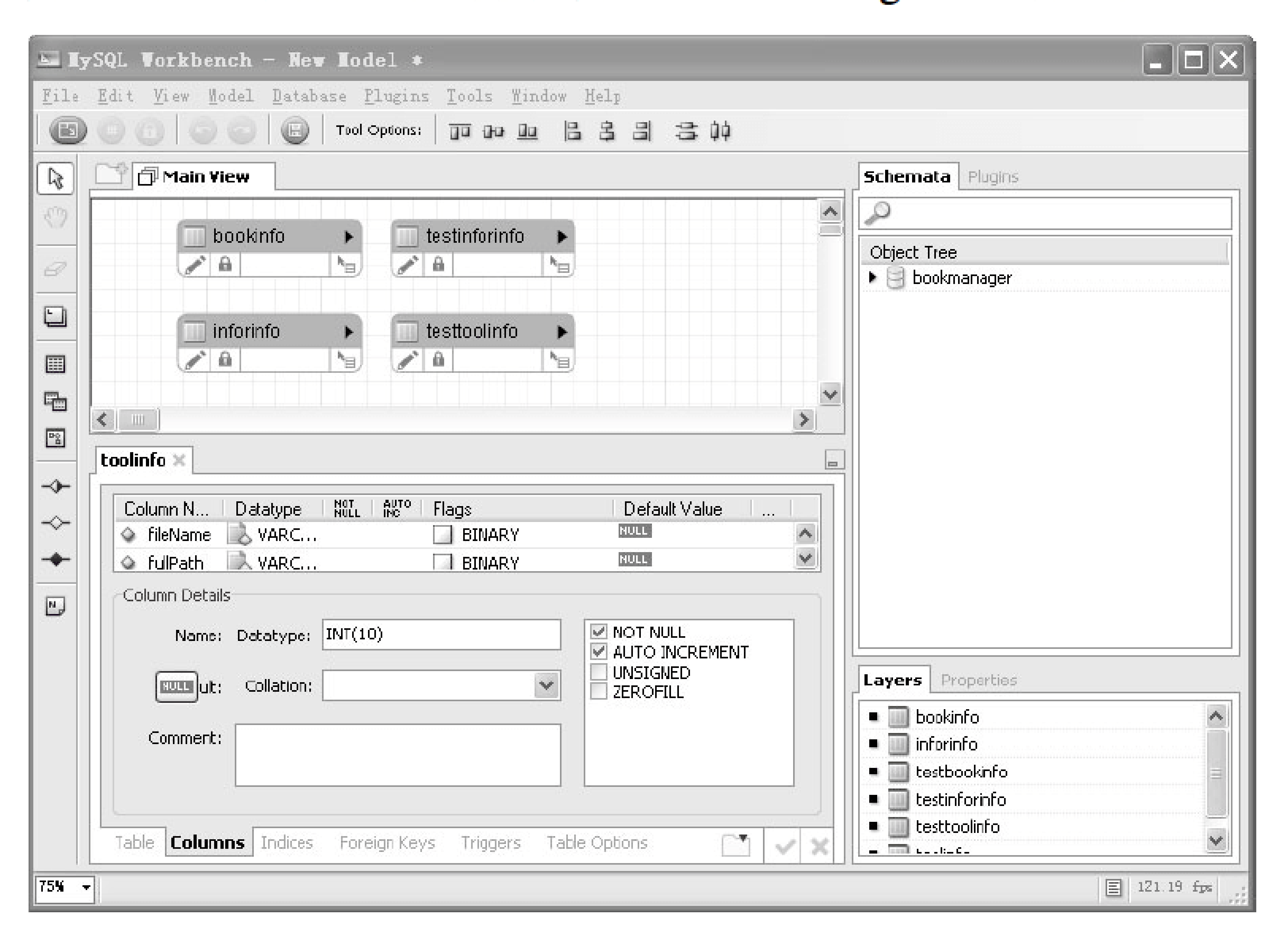


图 2-24　MySQL Workbench 工作界面

2.4　MySQL 的使用

本节将介绍 MySQL 客户端程序和服务端程序的基本使用方法。

2.4.1　MySQL 的基本使用

如果掌握了 MySQL 自带的命令式客户端程序的使用，那么也就能掌握 MySQL 的 GUI 工具。因此，虽然有 MySQL 的各种 GUI 工具可以方便我们的操作，但是掌握 MySQL 的命令行客户端是使用 MySQL 的基本能力之一。所以，这里主要介绍 MySQL 命令行客户端程序的使用。

本节的目的在于将读者快速带入 MySQL 世界，能够使用 MySQL 的基本功能。在后续章节中，会对 MySQL 命令行客户端作更详细的介绍。

在安装好 MySQL 后，可以在命令提示符下，输入相关的 MySQL 命令，对 MySQL 进行操作。打开命令提示符后，输入命令“MySQL-help”可以显示 MySQL 的帮助信息。

为了连接 MySQL 服务器，需要一台主机名、用户名和密码。连接的命令格式为：

mysql -h *host* -u *user* -p

mysql 命令行客户端界面如图 2-25 所示。

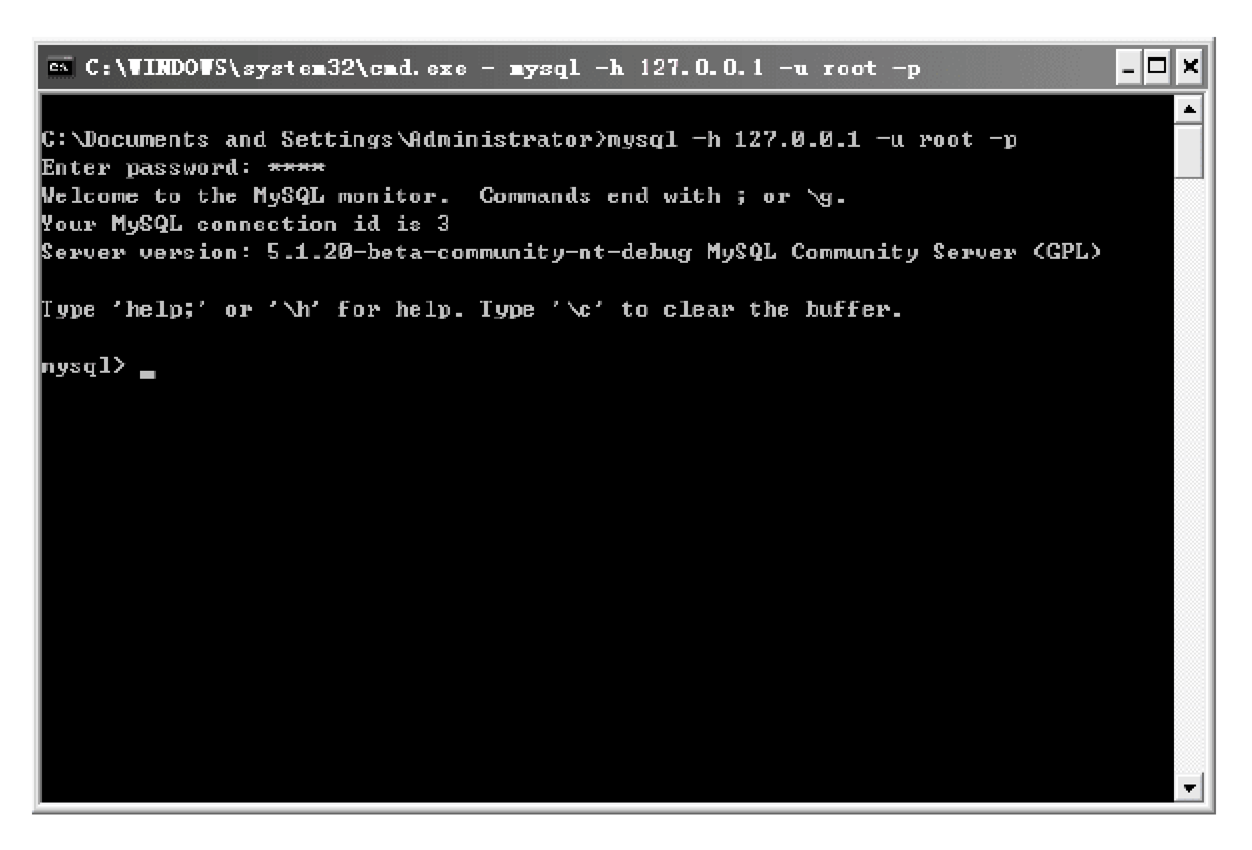


图 2-25　mysql 命令行客户端界面

Enter Password 提示输入密码。输入正确后，即可连接到 MySQL 服务器。此时，进入 MySQL 客户端的工作状态，以“mysql >”为提示符。如果要断开与 MySQL 服务器的连接，使用命令 QUIT 即可。

一个简单的 SQL 命令，可以显示当前 MySQL 的版本。

```
mysql> SELECT VERSION(), CURRENT_DATE;
```

这里需要说明几点：

（1）一个命令通常由 SQL 语句组成，随后跟着一个分号，对于这些命令，只有遇到分号结束时，才会被执行（有一些例外不需要分号。QUIT 是一个例子）。

（2）MySQL 用表格（行和列）方式显示查询输出。第一行包含列的标签，随后的行是查询结果。通常，列标签是取自数据库表的列的名字。如果正在检索一个表达式而非表列的值（如刚才的例子），MySQL 用表达式本身标记列。

（3）MySQL 显示返回了多少行，以及查询花了多长时间，它给用户提供服务器性能的一个大致概念。因为它们表示时钟时间（不是 CPU 或机器时间），并且因为它们受到诸如服务器负载和网络延时的影响，因此这些值是不精确的。

（4）MySQL 的查询不区分大小写。

以下是一个简单的 SQL 命令用于进行一个科学计算。

```
mysql> select pi()*2,4+4/3;
+----------+--------+
| pi()*2   | 4+4/3 |
+----------+--------+
| 6.283185 | 5.3333|
+----------+--------+
1 row in set (0.01 sec)
```

可以在一行写入多个命令，以分号分隔，遇到回车时一起执行。如：

```
mysql> SELECT VERSION(); SELECT NOW();
```

这个命令相当于先执行“SELECT VERSION();”，执行完毕后，再执行“SELECT NOW();”。

同样，如果命令很长，不必在一行内写完所有的命令。如一个取得当前用户和当前日期的命令：

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
```

等价于命令：

```
mysql>SELECT USER(),CURRENT_DATE;
```

可以看到，MySQL 命令解析器都以分号为分隔符，而不是以回车符去解析各个命令。

1．创建和使用数据库

使用 MySQL 的 Show 命令可以查看当前服务器上存在的数据库。

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| bookmanager        |
| javabs             |
| mysql              |
| test               |
+--------------------+
5 rows in set (0.00 sec)
```

不同的 MySQL 服务器上通常有不同的数据库。但通常都有一个 mysql 数据库，它描述了用户的访问权限。Test 数据库通常是让用户初试身手的地方。

如果 test 数据库存在，并且需要访问它，可以使用 USE 命令。使用 USE 命令后，当前的数据库就会切换，之后对服务器发出的命令都会在当前的数据库上进行。

```
mysql> USE test
Database changed
```

USE 命令类似 QUIT 命令，不需要分号，但即使加上分号，也不会有错。还有一点值得注意，USE 命令只能在单行使用。

如果需要创建自己的数据库，可以使用 Create Database 命令。以下命令建立一个名为 javabs 的数据库。

```
mysql> create database javabs;
Query OK, 1 row affected (0.00 sec)
```

创建数据库并不意味着已经选定它们，必须使用 USE 命令选定需要操作的数据库。

```
mysql> use javabs
Database changed
```

除了使用 USE 命令选定数据库外，也可以在登录服务器时指定使用的数据库。如：

mysql -h *127.0.0.1* -u root --p javabs

【特别提示】javabs 不是登录密码，而是需要使用的数据库名称。如果要在-p 选项中指定密码，不应该使用空格，即应该使用-pmypassword，而不是-p mypassword，如 mysql -h 127.0.0.1 -u root --proot javabs。此时将会以用户名 root、密码 root 登录 mysql 服务器，并同时使用 javabs 数据库。但一般不推荐使用这种方式指定登录密码，因为这样容易导致密码被窃取。

2. 创建和使用表

创建并指定一个数据库后，即可对该数据库进行操作，如 test 数据库。

```
mysql> use test;
Database changed
mysql> show tables;
Empty set (0.01 sec)
```

以上命令显示了 test 数据库中所有的表，因为 test 数据库是空的，因此没有显示任何信息。此时，需要在 test 数据库中新建一些表。以一个客户为例，有记录型：

客户（姓名，性别，住址，电话）

建立一张数据库表，存放客户的信息。

```
mysql> CREATE TABLE custom (
    ->   name VARCHAR(45) NOT NULL,
    ->   sex TINYINT UNSIGNED NOT NULL,
    ->   address VARCHAR(60) NOT NULL,
    ->   tel VARCHAR(45) NOT NULL,
    ->   PRIMARY KEY (`name`)
    -> ) ENGINE = InnoDB;
Query OK, 0 rows affected (0.09 sec)
```

此时，数据库中就已经新建了一张 custom 表。使用 show tables 命令进行查看。

```
mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| custom         |
+----------------+
1 row in set (0.00 sec)
```

可以看到，custom 表已经成功建立了。如果需要查看 custom 表的详细信息，可以使用 DESCRIBE 命令。如：

```
mysql> DESCRIBE custom;
+---------+---------------------+------+-----+---------+-------+
| Field   | Type                | Null | Key | Default | Extra |
+---------+---------------------+------+-----+---------+-------+
| name    | varchar(45)         | NO   | PRI |         |       |
| sex     | tinyint(3) unsigned | NO   |     |         |       |
| address | varchar(60)         | NO   |     |         |       |
| tel     | varchar(45)         | NO   |     |         |       |
+---------+---------------------+------+-----+---------+-------+
4 rows in set (0.13 sec)
```

通过这个命令，可以查看 custom 表各个字段的详细情况。

通过使用 insert 命令可以向这个空表添加数据。

```
mysql> insert into custom
    -> values('小张',0,'文一路','88320625');
Query OK, 1 row affected (0.08 sec)
```

数据添加完成后，可以使用 select 命令查看表中所有的数据。

```
mysql> select * from custom;
+------+-----+---------+----------+
| name | sex | address | tel      |
+------+-----+---------+----------+
| 小张 |   0 | 文一路  | 88320625 |
+------+-----+---------+----------+
1 row in set (0.05 sec)
```

有关 select、insert 命令的详细使用方法，可以参考第 3 章 SQL 基础知识。

2.4.2　MySQL 客户端程序

MySQL 客户端位于 MySQL 安装目录下的 bin 文件夹中。

1．常用命令行参数

- --help，-?：显示帮助消息并退出。
- --database=db_name，-D db_name：要使用的数据库，主要在选项文件中有用。
- --execute =name，-e name：执行一个命令，并且退出 mysql。
- --force，-f：即使出现一个 SQL 错误仍继续。
- --host=host_name，-h host_name：连接给定主机上的 MySQL 服务器。
- --html，-H：产生 HTML 输出。
- --password[=password]，-p[password]：当连接服务器时使用的密码。如果使用短选项形式（-p），选项和密码之间不能有空格。如果在命令行中--password 或-p 选项后面没有密码值，则提示输入一个密码。
- --port=port_num，-P port_num：用于连接的 TCP/IP 端口号。
- --user=user_name，-u user_name：当连接服务器时 MySQL 使用的用户名。

2．mysql 命令

mysql 将发出的 SQL 语句发送到待执行的服务器。还有一系列命令 mysql 可以自己解释。要查看这些命令，在 mysql>提示下输入“help”或“\h”：

```
mysql> help
List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?          (\?) Synonym for 'help'.
clear      (\c) Clear command.
connect    (\r) Reconnect to the server. Optional arguments are db and host.
delimiter  (\d) Set statement delimiter. NOTE: Takes the rest of the line
```

```
                 as new delimiter.
ego          (\G) Send command to mysql server, display result vertically.
exit         (\q) Exit mysql. Same as quit.
go           (\g) Send command to mysql server.
help         (\h) Display this help.
notee        (\t) Don't write into outfile.
print        (\p) Print current command.
prompt       (\R) Change your mysql prompt.
quit         (\q) Quit mysql.
rehash       (\#) Rebuild completion hash.
source       (\.) Execute an SQL script file. Takes a file name as an argument.
status       (\s) Get status information from the server.
tee          (\T) Set outfile [to_outfile].Append everything into given outfile.
use          (\u) Use another database. Takes database name as argument.
charset      (\C) Switch to another charset. Might be needed for processing
                  binlog with multi-byte charsets.
warnings     (\W) Show warnings after every statement.
nowarning    (\w) Don't show warnings after every statement.
```

Help 命令列出一个 MySQL 客户端程序可以解析的一些命令，并对其作出了解释。一些常用的命令如下。

- clear(\c)：取消当前正需要执行的命令。如：

```
mysql> select * from custom\c
```

此时，SQL 语句 select * from custom 将不被执行。

- Quit(\q)：退出 MySQL 客户端。
- Source(\.)：指定并执行一个 sql 文件。
- Use(\u)：指定使用一个数据库。

3. mysql 使用实例

- 登录 mysql 服务器：mysql –h 127.0.0.1 -u root –proot。
- 登录 mysql 服务器，并进入 mysql 数据库：mysql –h 127.0.0.1 -u root -proot -D mysql。
- 登录 mysql 服务器，并执行 binlog.sql 中的 sql 语句，然后退出 mysql：mysql -u root -proot -e "source d:/binlog.sql"。

2.4.3 MySQL 服务端程序

mysqld.exe 是 MySQL 的服务端程序。在 Windows 下 mysqld-nt.exe 是 mysqld.exe 的优化版本。因此在 Windows 下使用 mysqld 和 mysqld-nt 功能是相同的。它们位于 MySQL 安装目录下的 bin 文件夹中。运行 mysqld 时可以指定各种参数来配置 MySQL 的各项参数。使用命令 mysqld --verbose --help 可以查看 MySQL 支持的所有命令行参数。由于 MySQL 支持参数很多，往往不便于查看。可以使用输出重定向命令将它输入一个文本文件中。如在命令提示符下执行如下命令：

```
mysqld --verbose --help > c:\mysqlHelp.txt
```

此时，有关 mysqld 的帮助信息会保存在文件 c:\mysqlHelp.txt 中，这些信息包括 mysqld 的启动参数和环境变量。

一般来说，不建议在命令行指定 mysql 的运行参数。推荐使用 MySQL 的配置文件进行配置。

一个典型的通过命令行启动 mysql 服务器的命令为：

```
mysqld-nt --defaults-file="D:\tools\MySQL5.1\my.ini"
```

my.ini 为 mysqld 的默认配置文件，位于 MySQL 的安装目录下。通过参数--defaults-file 进行指定。当然也可以指定其他文件作为 MySQL 的配置文件。在 Windows 下，如果在安装时将 MySQL 安装为系统服务，则不需要在命令行启动 MySQL，可以在控制面板->管理工具->服务选项中找到 MySQL 的服务名称，如图 2-26 所示。在服务管理中启动或者停止它。

查看 MySQL 服务的属性，如图 2-27 所示，在可执行文件的路径一栏中可以看到，该服务的启动命令正是：

```
mysqld-nt --defaults-file="D:\tools\MySQL5.1\my.ini"
```

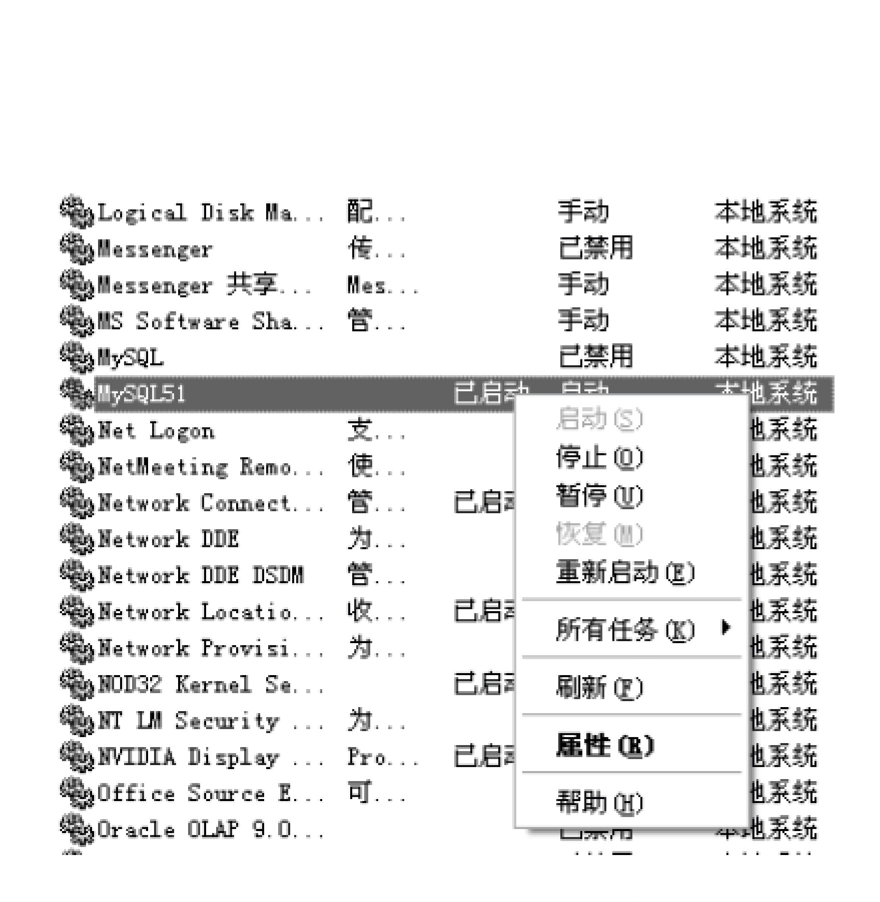


图 2-26　mysql 服务



图 2-27　mysql 服务属性

在安装 MySQL 时，可以直接将 MySQL 安装为 Windows 服务。也可以手动安装，或者拆卸 MySQL 服务。

在拆卸 MySQL 服务时，首先需要停止 MySQL 服务的运行。然后执行命令：

```
mysqld -remove mysql51
```

mysql51 为笔者计算机中 MySQL 的服务名，根据实际情况，每台计算机中的 MySQL 服务名称可能不同，可由用户自定义。mysqld -remove mysql51 执行成功后，mysql51 服务就被拆卸了。

安装 MySQL 为服务的命令是：mysqld -install mysql51。mysql51 是服务名称，可以根据自己的喜好指定。运行成功后，系统中就多了名为 mysql51 的服务。

通过修改 mysqld 的默认配置文件 my.ini 可以更改 MySQL 的启动参数。my.ini 文件中[mysqld]段可以指定 mysqld 服务器的一系列启动参数和环境变量。

在使用 MySQL 客户端登录服务器后，使用命令“show Variables;”可以查看服务器运行时的所有环境变量。

以下简单介绍 mysqld 服务器的一些常用启动选项：

- --help，-?：显示简短的帮助消息并退出。使用--verbose 和--help 选项来查看全部内容。
- --ansi：使用标准（ANSI）SQL 语法代替 MySQL 语法。使用--sql-mode 选项可以更精确控制服务器 SQL 模式。
- --bind-address=IP：待绑定的 IP 地址。
- --console：即使指定了--log-error 也将错误日志消息写入 stderr 和 stdout。在 Windows 中，如果使用该选项，mysqld 不会关闭控制台窗口。
- --character-sets-dir=path：字符集安装的目录。
- --character-set-server=charset：使用 charset 作为默认服务器字符集。
- --datadir=path，-h path：数据目录的路径。
- --init-file=file：启动时从该文件读 SQL 语句。每个语句必须在同一行中并且不应包括注释。
- ---log[=file]，-l [file]：日志连接和对文件的查询。如果不指定文件名，MySQL 使用 host_name.log 作为文件名。
- --log-bin=[file]：二进制日志文件。将更改数据的所有查询记入该文件，用于备份和复制。如果不指定，MySQL 使用 host_name-bin 作为日志文件基本名。
- --skip-bdb：禁用 BDB 存储引擎。这样可以节省内存，并可能加速某些操作。如果需要 BDB 表则不要使用该选项。
- --skip-grant-tables：该选项使服务器不使用权限系统。该权限允许访问服务器的用户不受限制地访问所有数据库。可以从系统外壳命令行执行 mysqladmin flush-privileges 或 mysqladmin reload 命令，或执行 MySQL FLUSH PRIVILEGES 语句让运行的服务器重新开始使用授权表。
- --skip-innodb：禁用 InnoDB 存储引擎。这样可以节省内存，并可能加速某些操作。如果需要 innodb 表则不要使用该选项。
- --sql-mode=value[,value[,value...]]：将 MySQL 设置为 SQL 模式。

主要的 sql_mode 值为：

- ANSI

更改语法和行为，使其更符合标准 SQL。

- STRICT_TRANS_TABLES

如果不能将给定的值插入到事务表中，则放弃该语句。对于非事务表，如果值出现在单行语句或多行语句的第一行，则放弃该语句；如果错误出现在后续行，那么调整后续行的值，使之插入或更新到数据库，同时给出警告信息，而不抛出错误。

- TRADITIONAL

MySQL 的行为像“传统”SQL 数据库系统。当在列中插入不正确的值时“给出错误而

不是警告”。一旦发现错误该模式立即放弃 INSERT/UPDATE。如果使用非事务存储引擎，这种方式不能保证数据的完整性，因为遇到错误后不会进行回滚操作，结果是更新“只进行了一部分”。

- --transaction-isolation=level：设置默认事务隔离级别，可以是 READ-UNCOMMITTED、READ-COMMITTEE、REPEATABLE-READ 或 SERIALIZABLE。

2.5　MySQL 工具程序的使用

在 MySQL 的安装目录的 bin 文件夹下，有 MySQL 的相关使用程序。这些程序包括：

- mysqladmin 是用于管理功能的客户程序。
- mysqlcheck 执行表维护操作。
- mysqldump 和 mysqlhotcopy 负责数据库备份。
- mysqlshow 显示信息数据库和表的相关信息。
- myisamchk 执行表维护操作。
- myisampack 产生压缩、只读的表。
- mysqlbinlog 是处理二进制日志文件的实用工具。
- perror 显示错误代码的含义。

在 Windows 下，要调用以上程序，只要将 mysql 安装目录下的 bin 目录加入到 PATH 环境变量中，即可在命令提示符下进行使用。在 Windows XP 中，可以在开始->运行菜单中执行 cmd 命令进入命令提示符。

2.5.1　mysqladmin

mysqladmin 是一个用于管理 MySQL 服务器的客户端程序。它可以检查服务器的配置和当前的状态、创建并删除数据库等。mysqlamdin 连接到服务器的方法和 MySQL 命令行客户端程序是一样的。mysqladmin 支持以下常用命令。

- create db_name：创建一个名为 db_name 的新数据库。
- debug：告诉服务器向错误日志写入调试信息。
- drop db_name：删除名为 db_name 的数据库和所有表。
- extended-status：显示服务器状态变量及其值。
- flush-hosts：刷新主机缓存中的所有信息。
- flush-logs：刷新所有日志。
- flush-privileges：重载授权表（类似 reload）。
- flush-status：清除状态变量。
- flush-tables：刷新所有表。
- flush-threads：刷新线程缓存。
- kill id，id，...：“杀掉”服务器线程。

- password new-password：设置一个新密码。将用 mysqladmin 连接服务器使用的账户的密码更改为 new-password。如果 new-password 包含空格或其他命令解释符的特殊字符，需要用引号将它引起来。在 Windows 中，一定要使用双引号而不要用单引号；单引号不会从密码中剥离出来，而是解释为密码的一部分。例如，可以使用 mysqladmin password "my new password"，而不能使用 mysqladmin password 'my new password'。
- Ping：检查服务器是否仍活动。如果服务器运行 mysqladmin 返回状态 0，如果不运行返回 1。即使出现错误例如 Access denied 也为 0，因为这说明服务器在运行但拒绝了连接，与服务器不在运行不同。
- Processlist：显示活动服务器线程的列表。类似 SHOW PROCESSLIST 语句的输出。如果给出了--verbose 选项，输出类似 SHOW FULL PROCESSLIST 的语句。
- Reload：重载授权表。
- Refresh：刷新所有表并关闭和打开日志文件。
- Shutdown：停止服务器。
- start-slave：开始从服务器上的复制。
- Status：显示服务器状态消息。
- stop-slave：停止从服务器上的复制。
- Variables：显示服务器系统变量及其值。
- Version：显示服务器的版本信息。

所有命令可以简化为任何唯一的前缀，即命令 Version 和 ver 是等价的，命令 Variables 和 var 是等价的。

以下示例中，默认连接本机数据库，如果要连接远程数据库，需要用-h 选项指定数据库主机名称或 IP。-u 指定登录数据库的用户名，-p 指定用户密码。

例 2-1　新建一个名为 javabs 的数据库：

```
mysqladmin  -u root -proot create javabs
```

例 2-2　删除名为 javabs 的数据库及其表：

```
mysqladmin  -u root -proot drop javabs
Dropping the database is potentially a very bad thing to do.
Any data stored in the database will be destroyed.
Do you really want to drop the 'javabs' database [y/N] y
Database "javabs" dropped
```

例 2-3　显示服务器环境变量：

```
mysqladmin  -u root -proot Variables 或者 mysqladmin  -u root -proot Var
```

【特别提示】由于服务器的环境变量很多，全部打印出来后很难查找到自己需要的变量。为此，可以使用 find 命令进行查找（在 UNIX 或 linux 下使用 grep 命令），例如现在要查看环境变量 long_query_time 的值，可以使用命令: mysqladmin -u

root -proot var|find "long_query_time"，该命令的含义是，将 mysqladmin -u root -proot var 的输出作为 find 命令的输入，在众多环境变量中可以快速查找定位到 long_query_time 变量。有关 find 函数的使用方法，可以使用命令 find/?查看。

例 2-4　查看数据库是否在运行：

```
mysqladmin  -u root -proot ping
mysqld is alive
```

例 2-5　显示服务器状态信息：

```
mysqladmin  -u root -proot status
Uptime: 2857  Threads: 2  Questions: 24  Slow queries: 0  Opens: 16  Flush
tables: 1  Open tables: 2  Queries per second avg: 0.8
```

- Uptime：MySQL 服务器已经运行的秒数。
- Threads：活动线程（客户）的数目。
- Questions：服务器启动以来客户的问题（查询）数目。
- Slow queries：执行时间超过 long_query_time 秒的查询的数量。
- Opens：服务器已经打开的数据库表的数量。
- Flush tables：服务器已经执行的 flush、…、refresh 和 reload 命令的数量。
- Open tables：目前打开的表的数量。
- Queries per second avg：平均每秒的查询数量。

例 2-6　显示服务器线程列表：

```
mysqladmin  -u root -proot processlist
```

或者

```
mysqladmin  -u root -proot proc
```

2.5.2　mysqlcheck

mysqlcheck 客户端可以检查和修复 MyISAM 表。它还可以优化和分析表。

mysqlcheck 的功能类似 myisamchk，但其工作不同。主要差别是当 mysqld 服务器在运行时必须使用 mysqlcheck，而 myisamchk 应用于服务器没有运行时。使用 mysqlcheck 的好处是不需要停止服务器来检查或修复表。

mysqlcheck 连接服务器的方法和 MySQL 客户端程序是一样的。即使用-u 指定用户名，-p 指定密码，-h 指定 mysql 服务器主机名称。

有 3 种方式可以调用 mysqlcheck：

```
mysqlcheck[options] db_name [tables]
mysqlcheck[options] ---database DB1 [DB2 DB3...]
mysqlcheck[options] --all--database
```

mysqlcheck 支持以下常用选项。

- --help，-？：显示帮助消息并退出。
- --all--database，-A：检查所有数据库中的所有表。与使用--database 选项相同，在命令行中命名所有数据库。
- --analyze，-a：分析表。
- --auto-repair：如果某个被检查的表破坏了，自动修复它。检查完所有表后自动进行所有需要的修复。
- --check，-c：检查表的错误。
- --check-only-changed，-C：只检查上次检查以来已经更改的或没有正确关闭的表。
- --database，-B：处理数据库中命名的所有表。使用该选项，所有字名参量被看作数据库名，而不是表名。
- --fast，-F：只检查没有正确关闭的表。
- --force，-f：即使出现 SQL 错误也继续。
- --medium-check，-m：执行比--extended 操作更快的检查。只能发现 99.99%的错误，在大多数情况下这已经足够了。
- --optimize，-o：优化表。
- --port=port_num，-P port_num：用于连接的 TCP/IP 端口号。
- --repair，-r：执行可以修复大部分问题的修复，只是唯一值不唯一时不能修复。
- --tables：覆盖---database 或-B 选项。选项后面的所有参量被视为表名。
- --version，-V：显示版本信息并退出。

以下示例中，默认连接本机数据库，如果要连接远程数据库，需要用-h 选项指定数据库主机名称或 IP，-u 指定登录数据库的用户名 ，-p 指定用户密码。

例 2-7　检查 mysql 数据库中所有的表：

```
mysqlcheck -u root -proot -c -B mysql
```

例 2-8　检查 mysql 数据库中的 user 表：

```
mysqlcheck -u root -proot -c mysql user
```

例 2-9　检查所有数据库中的所有表：

```
mysqlcheck -u root -proot -c -A
```

例 2-10　分析 mysql 数据库中的 user 表：

```
mysqlcheck -u root -proot -a mysql user
```

例 2-11　修复 mysql 数据库中的 user 表：

```
mysqlcheck -u root -proot -r mysql user
```

2.5.3　mysqldump

mysqldump 客户端可用来转储数据库或搜集数据库进行备份或将数据转移到另一个 SQL 服务器（不一定是一个 MySQL 服务器）。转储包含创建表和/或装载表的 SQL 语句。

有 3 种方式可以调用 mysqldump：

```
mysqldump [options] db_name [tables]
mysqldump [options] --database DB1 [DB2 DB3...]
mysqldump [options] --all--database
```

如果没有指定任何表或使用了--database 或--all--database 选项，则转储整个数据库。

常用的 mysqldump 选项如下。

- --help，-？：显示帮助消息并退出。
- --add-drop--database：在每个 CREATE DATABASE 语句前添加 DROP DATABASE 语句。
- --add-drop-tables：在每个 CREATE TABLE 语句前添加 DROP TABLE 语句。
- --add-locking：用 LOCK TABLES 和 UNLOCK TABLES 语句引用每个表转储。重载转储文件时插入得更快。
- --all--database，-A：转储所有数据库中的所有表。与使用--database 选项相同，在命令行中命名所有数据库。
- --allow-keywords：允许创建关键字列名。应在每个列名前面加上表名前缀。
- --force，-f：在表转储过程中，即使出现 SQL 错误也继续。
- --no-data，-d：不写表的任何行信息。如果只想转储表的结构这很有用。
- --port=port_num，-P port_num：用于连接的 TCP/IP 端口号。
- --quick，-q：该选项用于转储大的表。它强制 mysqldump 从服务器一次一行地检索表中的行，而不是检索所有行并在输出前将它缓存到内存中。
- --triggers：为每个转储的表转储触发器。该选项默认启用；用--skip-triggers 禁用它。
- --where='where-condition'，-w 'where-condition'：只转储给定的 WHERE 条件选择的记录。注意，如果条件包含命令解释符专用空格或字符，一定要将条件引用起来。
- --xml，-X：将转储输出写成 XML。

例 2-12　将 mysql 数据库中的所有表保存到文件 C:\mysql.sql 中：

```
mysqldump -u root -proot --opt mysql>c:\mysql.sql
```

例 2-13　将 mysql 数据库中的 user 表保存到文件 C:\mysql–user.sql 中：

```
mysqldump -u root -proot mysql user>c:\mysql-user.sql
```

例 2-14　只保存 mysql 数据库中 user 表的结构：

```
mysqldump -u root -proot -d mysql user>c:\mysql-user.sql
```

例 2-15　只导出 mysql 数据库 user 表中的 root 用户：

```
mysqldump -u root -proot --where=user='root' mysql user>c:\mysql-user.sql
```

例 2-16　将备份的 mysql 数据库文件导入 mysql 数据库：

```
mysqldump -u root -proot mysql<c:\mysql.sql
```

2.5.4　mysqlshow

mysqlshow 客户可用来很快地查找存在哪些数据库，数据库中的表、表中的列或索引。它连接数据库服务器的方式和 mysql 命令行客户端相同，通过-u,-p,-h 来指定用户名、密码和服务器主机名。

mysqlshow 的调用方式为：

```
mysqlshow[选项] [db_name [tbl_name [col_name]]]
```

如果没有给出数据库，显示所有匹配的数据库；如果没有给出表，显示数据库中所有匹配的表；如果没有给出列，显示表中所有匹配的列和列类型。

mysqlshow 支持以下常用选项。

- ---help，-？：显示一个帮助消息并退出。
- --port=port_num，-P port_num：连接时使用的 TCP/IP 端口号。
- --status，-i：显示关于每个表的额外信息。
- --verbose，-v：冗长模式。打印出程序操作的详细信息。该选项可以多次使用以便增加信息总量。
- --version，-V：显示版本信息并退出。

例 2-17　查看 mysql 服务器中的所有数据库：

```
mysqlshow -u root -proot
```

例 2-18　查看 mysql 数据库中的所有表：

```
mysqlshow -u root -proot mysql
```

例 2-19　查看 mysql 数据库中 user 表的信息：

```
mysqlshow -u root -proot mysql user
```

例 2-20　查看 mysql 数据库中 user 表的额外信息：

```
mysqlshow -u root -proot -i mysql user
```

2.5.5　myisamchk

可以使用 myisamchk 实用程序来获得有关数据库表的信息或检查、修复、优化它们。

myisamchk 适用 MyISAM 表（对应.MYI 和.MYD 文件的表）。

myisamchk 的调用方法是：

```
myisamchk [options] tbl_name ...
```

常用的选项介绍如下。

- --help，-?：显示帮助消息并退出。
- --silent，-s：沉默模式。仅当发生错误时写输出。能使用-s 两次（-ss）使 myisamchk 沉默。
- --verbose，-v：冗长模式。打印更多的信息。这能与-d 和-e 一起使用。为了更冗长，使用-v 多次（-vv，-vvv）。
- --version，-V：显示版本信息并退出。
- --check，-c：检查表的错误。如果不明确指定操作类型选项，这就是默认操作。
- --check-only-changed，-C：只检查上次检查后有变更的表。
- --extend-check，-e：非常仔细地检查表。如果表有许多索引将会相当慢。该选项只能用于极端情况。一般情况下，可以使用 myisamchk 或 myisamchk --medium-check 来确定表内是否有错误。如果使用了--extend-check 并且有充分的内存，将 key_buffer_size 变量设置为较大的值，可以使修复操作运行得更快。
- --fast，-F：只检查没有正确关闭的表。
- --force，-f：如果 myisamchk 发现表内有任何错误，则自动进行修复。维护类型与--repair 或-r 选项指定的相同。
- --backup，-B：将.MYD 文件备份为 file_name-time.BAK。
- --recover，-r：可以修复几乎所有一切问题，除非唯一的键不唯一时（对于 MyISAM 表，这是几乎不可能的情况）。如果想要恢复表，这是首先要尝试的选项。如果 myisamchk 报告表不能用-r 恢复，则只能尝试-o。不过，如果-r 不能恢复数据，那么表的损坏情况可能相当严重。如果计算机有较大的内存，应尝试增加 sort_buffer_size 的值。
- --safe-recover，-o：使用旧的恢复方法按顺序读取所有行，并根据找到的行更新所有索引树。这比-r 慢些，但是能处理-r 不能处理的情况。该恢复方法使用的硬盘空间比-r 少。一般情况下，应首先用-r 维修，如果-r 失败则用-o。
- --analyze，-a：分析键值的分布。这通过让联结优化器更好地选择表应该以什么次序联结和应该使用哪个键来改进联结性能。要想获取分布相关信息，可以使用 myisamchk --description --verbose tbl_name 命令或 SHOW KEYS FROM tbl_name 语句。

例 2-21　检查 mysql 数据库的 user 表：

```
myisamchk user.myi
```

例 2-22　彻底检查 mysql 数据库的 user 表：

```
myisamchk -e user.myi
```

例 2-23　快速恢复 mysql 数据库 user 表：

```
myisamchk -r -q user
```

2.5.6　myisampack

myisampack 工具可以压缩 MyISAM 表。myisampack 分别压缩表中的每一列。通常，myisampack 可以将数据文件压缩到 40%~70%。

使用 myisampack 时，需要注意以下几个问题：

（1）如果 mysqld 服务器使用--skip-external-locking 选项启动，那么在压缩过程中表可能被更新，这种情况下不应该使用 myisampack 命令。

（2）表压缩后，它变为只读。不能进行修改操作。

（3）myisampack 可以压缩 BLOB 或 TEXT 列。旧版本 ISAM 表的 pack_isam 程序不可以。

myisampack 的调用格式为：

```
myisampack [options] filename ...
```

myisampack 支持的常用选项介绍如下。

- --help，-？：显示帮助消息并退出。
- --backup，-b：使用 tbl_name.OLD 名备份表数据文件。
- --silent，-s：沉默模式。只有发生错误时才写输出。
- --verbose，-v：冗长模式。写压缩操作过程相关信息和其结果。
- --version，-V：显示版本信息并退出。

例 2-24　压缩 test 数据库中的 custom 表：

```
myisampack custom
```

【特别提示】压缩表后，需要使用 myisamchk –rq 命令对压缩表重新创建索引。如使用命令 myisamchk -rq --sort-index --analyze custom。压缩表并不会被数据库立即使用，需要使用 mysqladmin -u root -proot flush-tables 命令强行让数据库更新表。其中-u 指定用户名，-p 指定密码。

例 2-25　解压 custom 表：

```
myisamchk --unpack custom
```

同样，解压后的表也需要使用 mysqladmin -u root -proot flush-tables 将新表更新到数据库中。

2.5.7　mysqlbinlog

MySQL 数据库采用二进制文件格式的方式记录数据库的日志。mysqlbinlog 用来读取这

些二进制日志。

二进制日志包含了所有更新了数据或者已经潜在更新了数据（例如，没有匹配任何行的一个 DELETE）的所有语句。语句以“事件”的形式保存，它描述数据更改。二进制不包含没有修改任何数据的语句。如果想要记录所有语句（例如，为了识别有问题的查询），应使用一般查询日志。

运行服务器时若启用二进制日志则性能大约慢 1%，但是这些性能损失却能大大提供服务器的可靠性，因此是非常值得的。

如果需要服务器进行二进制日志记录，需要在服务器启动时指定--log-bin[=*file_name*]选项。推荐编辑 my.ini 文件来指定该选项。在 mysql 的安装目录下，打开 my.ini 文件。在[mysqld]段下加入 log-bin=binlog。My.ini 文件可能如下所示：

```
[mysqld]
port=3306
basedir="D:/tools/MySQL5.1/"
datadir="D:/tools/MySQL5.1/Data/"
default-character-set=latin1
...
innodb_log_file_size=10M
innodb_thread_concurrency=8
flush_time=2000
log-bin=binlog
```

log-bin=binlog 指定了服务器启动时需要进行二进制日志的记录，并将二进制数据文件保存在 mysql 安装目录的 data 文件夹下，以 binlog.000001、binlog.000002、binlog.000003 等命名。每个日志的长度达到 max_binlog_size 后，就会自动新建一个日志文件。

mysqlbinlog 工具用于分析数据库的二进制日志，它支持以下常用选项。

- ---help，-？：显示帮助消息并退出。
- ---database=db_name，-d db_name：只列出该数据库的条目（只用本地日志）。
- --force-read，-f：使用该选项，如果 mysqlbinlog 读它不能识别的二进制日志事件，它会打印警告，忽略该事件并继续。没有该选项，如果 mysqlbinlog 读到此类事件则停止。
- --host=host_name，-h host_name：获取给定主机上的 MySQL 服务器的二进制日志。
- --password[=password]，-p[password]：当连接服务器时使用的密码。如果使用短选项形式（-p），选项和密码之间不能有空格。如果在命令行中--password 或-p 选项后面没有密码值，则提示输入一个密码。
- --port=port_num，-P port_num：用于连接远程服务器的 TCP/IP 端口号。
- --protocol={TCP | SOCKET | PIPE | -position ：使用的连接协议。
- --start-datetime=datetime：从二进制日志中第一个日期时间等于或晚于 datetime 参量的事件开始读取。datetime 值相对于运行 mysqlbinlog 的机器上的本地时区。该值格式应符合 DATETIME 或 TIMESTAMP 数据类型。
- --stop-datetime=datetime：从二进制日志中第一个日期时间等于或晚于 datetime 参

量的事件起停止读。关于 datetime 值的描述参见--start-datetime 选项。该选项可以帮助及时恢复。

- --start-position=N：从二进制日志中第一个位置等于 N 参量时的事件开始读。
- --stop-position=N：从二进制日志中第一个位置等于和大于 N 参量时的事件起停止读。
- --user=user_name，-u user_name：连接远程服务器时使用的 MySQL 用户名。
- --version，-V：显示版本信息并退出。

例 2-26　将日志文件的内容输入到文件 D:\logbin.txt：

```
mysqlbinlog binlog.000001>>D:\binlogtxt
```

例 2-27　重新执行日志中的所有操作：

```
mysqlbinlog binlog.000001|mysql -u root -proot
```

例 2-28　重新执行 2007-08-14 19:33:56 之前进行的所有操作：

```
mysqlbinlog --stop-datetime="2007-08-14 19:33:56" binlog.000001|mysql -u
root -proot
```

【特别提示】如果要同时执行多个日志文件，应该在一个数据库连接中处理它们。如以下处理方式是不正确的。

```
mysqlbinlog binlog.000001 | mysql -u root -proot
mysqlbinlog binlog.000002 | mysql -u root -proot
```

如果第一个日志文件包含一个 CREATE TEMPORARY TABLE 语句，第二个日志包含一个使用该临时表的语句，则会造成问题。当第一个 mysql 进程结束时，服务器撤销临时表；当第二个 mysql 进程想使用该表时，服务器报告“找不到临时表”。

因此，正确的处理方式应该是：

```
mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -proot
```

或者先将日志文件转化为 sql：

```
 mysqlbinlog binlog.000001 >  D:/ binlog.sql
 mysqlbinlog binlog.000002 >> D: / binlog.sql
 mysql -u root -proot -e "source d:/binlog.sql"
```

2.5.8　mysqlimport

mysqlimport 程序可以将文件中的数据导入到数据库中。mysqlimport 的使用语法如下：

```
mysqlimport [options] db_name textfile1 [textfile2 ...]
```

这里需要注意，文件的名称需要和要导入的表的名称相同。例如在本节示例中，导入 custom 表的文件名为 custom.txt。导入的文件可以是一个以 tab 符分割各个列，回车符分割

各个行的文本文件。

options 选项所支持的主要参数如下。

- ❑ ---help，-？：显示帮助消息并退出。
- ❑ --columns=column_list，-c column_list：该选项采用逗号分隔的列名作为其值。列名的顺序指示如何匹配数据文件列和表列。
- ❑ --delete，-d：导入文本文件前清空表。
- ❑ --force，-f：忽视错误。例如，如果某个文本文件的表不存在，继续处理其他文件。不使用--force，如果表不存在则 mysqlimport 退出。
- ❑ --host=host_name，-h host_name：将数据导入给定主机上的 MySQL 服务器。默认主机是 localhost。
- ❑ --ignore-lines=n：忽视数据文件的前 n 行。
- ❑ --password[=password]，-p[password]：当连接服务器时使用的密码。如果使用-p 形式，选项和密码之间不能有空格。如果在命令行中--password 或-p 选项后面没有密码值，则提示输入一个密码。
- ❑ --port=port_num，-P port_num：用于连接的 TCP/IP 端口号。

例 2-29　将 custom.txt 中的数据导入 test 数据库中的 custom 表中：

```
mysqlimport -h 127.0.0.1 -u root -proot test c:\Custom.txt
```

例 2-30　将 custom.txt 中的数据导入 test 数据库中的 custom 表中，导入前先清空表中的数据，并且指定文本中各个列的顺序为 name,address,sex,tel：

```
mysqlimport -h 127.0.0.1 -d -c name,address,sex,tel -u root -proot test
c:\Custom.txt
```

2.5.9　perror

perror 工具可以对错误进行描述说明。使用语法是：

```
perror [options] [ERRORCODE [ERRORCODE...]]
```

OPTIONS 选项可以用以下值。

- ❑ -?，--help：显示帮助信息。
- ❑ -s，--silent：只显示错误的描述。.
- ❑ -v，--verbose：打印错误代码和错误消息。
- ❑ #　-V，--version：打印版本信息。

例 2-31　显示错误码为 1、2、3 的 3 个错误信息：

```
perror 1 2 3
OS error code   1:  Operation not permitted
OS error code   2:  No such file or directory
OS error code   3:  No such process
```

2.6　小　　结

通过本章的阅读，读者应该对 MySQL 基本使用方法有所了解，能够开启和关闭 MySQL 服务器，熟悉 MySQL 服务器的常用启动参数。能够通过 MySQL 命令行，或者 GUI 工具进行基本的数据库操作，如建立、删除表，插入、更新、删除记录等。在阅读本章 2.5 节后，应该对 MySQL 的实用工具程序有所了解，能够通过这些工具管理、备份、检查 MySQL 数据库。

第 3 章　SQL 基础知识

SQL 全称是“结构化查询语言（Structured Query Language）”，由 Boyce 和 Chamberlin 于 1974 年首先提出。1975—1979 年 IBM 公司圣约瑟研究实验室为其关系数据库管理系统 SYSTEM R 实现了这种语言（当时称 SQUARE 语言）。SQL 语言简洁，功能强大，简单易学，被众多计算机公司和软件公司所采用。经各公司的不断修改、扩充和完善，如今，无论是 Oracle、Sybase、Informix、SQL Server、Ingres，还是 PostgreSQL、MySQL、mSQL 都支持 SQL 语言作为查询语言，SQL 语言最终发展成为关系数据库的标准语言。

3.1　SQL 语言基本知识

【特别提示】SQL 的正确发音在数据库交流中是有争论的，对于 SQL 标准，美国标准协会认为官方发音是“ess-queue-el”，即字母“S”、“Q”、“L”的发音。然而，许多数据库专家用行话“sequel”，读作/ˈsi:kju:/。读者读哪个音都可以，附带提一下，MySQL 参考指南中 MySQL 的发音为“my-ess-queue-ell”。

3.1.1　SQL 的历史

在 20 世纪 70 年代初，E.F.Codd 首先提出了关系模型。70 年代中期，IBM 公司在研制 SYSTEM R 关系数据库管理系统中研制了 SQL 语言，最早的 SQL 语言（叫做 SEQUEL2）是在 1976 年 11 月的 IBM Journal of R&D 上公布的。

1979 年 Oracle 公司首先提供商用的 SQL，IBM 公司在 DB2 和 SQL/DS 数据库系统中也实现了 SQL。

1986 年 10 月，美国国家标准学会（American National Standard Institute，ANSI）的数据库委员会 X3H2 批准采用 SQL 作为关系数据库管理系统的标准语言(简称 SQL-86 标准)。1987 年国际标准化组织（International Organization for Standardization，ISO）也通过了这一标准。

1989 年，ANSI 采纳在 ANSI X3.135-1989 报告中定义的关系数据库管理系统的 SQL 标准语言，简称 SQL-89 标准。此后 ANSI 不断修改和完善 SQL 标准，SQL:2006 是目前最新的标准，标准 SQL 各版本演变简表如表 3-1 所示。

目前，所有主要的关系数据库管理系统支持某些形式的 SQL 语言，大部分数据库遵守 ANSI SQL92 标准。

从 SQL:1999 开始，SQL 标准的个头就非常大，其内容可以说是包罗万象，已经不仅仅

限于 SQL 语言本身了。SQL:2003 包括以下 9 个部分。

- ISO/IEC 9075-1: Framework (SQL/Framework)。
- ISO/IEC 9075-2: Foundation (SQL/Foundation)。
- ISO/IEC 9075-3: Call Level Interface (SQL/CLI)。
- ISO/IEC 9075-4: Persistent Stored Modules (SQL/PSM)。
- ISO/IEC 9075-9: Management of External Data (SQL/MED)。
- ISO/IEC 9075-10: Object Language Bindings (SQL/OLB)。
- ISO/IEC 9075-11: Information and Definition Schemas (SQL/Schemata)。
- ISO/IEC 9075-13: Java Routines and Types Using the Java Programming Language (SQL/JRT)。
- ISO/IEC 9075-14: XML-Related Specifications (SQL/XML)。

【特别提示】负责具体制定工作的是 ISO 和 IEC 联合成立的一个技术委员会 JTC1/SC32，其全名是：Joint Technical Committee ISO/IEC JTC 1、Information technology、Subcommittee SC 32、Data management and interchange。

表 3-1　标准 SQL 各版本演变简表

年　份	标准名称	别　名
1986	SQL-86	ANSI×3.135-1986，ISO/IEC9075：1986，SQL-87
1989	SQL-89	ANSI×3.135-1989，ISO/IEC9075：1989，FIPS 127-1
1992	SQL-92	ANSI×3.135-1992，ISO/IEC9075：1992，SQL2，FIPS 127-2
1999	SQL:1999	ISO/IEC 9075:1999，SQL3
2003	SQL:2003	ISO/IEC 9075:2003，SQL4
2006	SQL:2006	ISO/IEC 9075:2006，SQL5

【特别提示】细心的读者能发现，从 SQL:1999 开始，标准简称中的短横线（-）被换成了冒号（:），而且标准制定的年份也改用 4 位数字了。前一个修改的原因是 ISO 标准习惯上采用冒号，ANSI 标准则一直采用短横线。后一个修改的原因是标准的命名也遇到了 2000 年问题。

3.1.2　SQL 的特点

SQL 语言之所以能够为用户和业界所接受，并成为国际标准，是因为它是一个综合的、功能极强同时又简洁易学的语言。SQL 语言集数据查询语言 DQL（Data Query Language SELECT）、数据操纵语言 DML（Data Manipulation Language）、数据定义语言 DDL（Data Definition Language）、数据控制语言 DCL（Data Control Language）于一体。主要特点包括以下几点：

1．非过程化语言

SQL 是一个非过程化的语言，因为它一次处理一个记录，对数据提供自动导航。SQL

允许用户在高层的数据结构上工作，而不对单个记录进行操作，可操作记录集。所有 SQL 语句接受集合作为输入，返回集合作为输出。SQL 的集合特性允许一条 SQL 语句的结果作为另一条 SQL 语句的输入。SQL 不要求用户指定对数据的存放方法。这种特性使用户更易集中精力于要得到的结果。所有 SQL 语句使用查询优化器，它是关系型数据库管理系统 RDBMS（Relational DataBase Management System）的一部分，由它决定对指定数据存取的最快速度的手段。查询优化器知道存在什么索引，哪儿使用合适，而用户从不需要知道表是否有索引，表有什么类型的索引。

2. 综合统一

SQL 可用于所有用户的 DB 活动模型，包括系统管理员、数据库管理员、应用程序员、决策支持系统人员及许多其他类型的终端用户。基本的 SQL 命令只需很少时间就能学会，最高级的命令在几天内便可掌握。SQL 为许多任务提供了命令，可以独立完成数据库生命周期中的全部活动，包括以下几个选项。

- ❑ 数据定义（Data definition）：SQL 可用于定义被存放数据的结构和组织，以及数据项之间的关系。
- ❑ 数据检索（Data retrieval）：SQL 能使用户或应用程序从数据库中检索数据并使用这些数据。
- ❑ 数据操纵（Data manipulation）：用户或应用程序通过 SQL 更改数据库，如增加新数据、删除旧数据、修改已存入的数据等。
- ❑ 存取控制（Access control）：SQL 可用来限制用户检索、增加和修改数据的权限，以保护所存储的数据不被非法存取。
- ❑ 数据共享（Data sharing）：SQL 可用于调整数据让并发用户共享，以保证用户之间彼此不受影响。
- ❑ 数据完整性（Data integrity）：SQL 能对数据库的完整性条件作出规定，以使其不会因为修改紊乱或系统出错而被破坏。

用户在数据库投入运行后，可根据需要随时逐步修改模式，不影响数据的运行，从而使系统具有良好的可扩展性。

另外，在关系模型中实体和实体间的联系均用关系表示，这种数据结构的单一性带来了数据操作符的统一，查询、插入、删除、修改等每一种操作都只需要一种操作符，从而克服了非关系系统由于信息表示方式的多样性带来的操作复杂性。

以前的数据库管理系统为上述各类操作提供单独的语言，而 SQL 将全部任务统一在一种语言中。

3. 以同一种语法结构提供多种使用方式

SQL 语言是自含式语言，又是嵌入式语言。它既可以独立地用于联机交互的使用方式，用户可以在终端键盘上直接输入 SQL 命令对数据库进行操作，又可以把 SQL 语句嵌入到高级语言（例如 C++、Java）程序中。不管是自含式还是嵌入式，SQL 语言的语法结构基本上是一致的。这种以统一的语法结构提供两种不同的使用方式的做法，提供了极大的灵活性与方便性。

由于所有主要的关系数据库管理系统都支持 SQL 语言，用户可将使用 SQL 的技能从一个 RDBMS 转到另一个。所有用 SQL 编写的程序都是可以移植的。

4. 面向集合的操作方式

SQL 采用集合操作方式，不仅操作对象、查找结果可以是元组的集合，而且一次插入、删除、更新操作的对象也可以是元组的集合。

【特别提示】可以把元组理解为在一张二维表格中的一行，它构成一个逻辑记录。二维表中的一列理解为一个域（又称字段），在一个相同的关键域中根据需要可有不同数量的元组（又称记录），以构成一个关系表，它是同一类数据的集合。

5. 语言简洁，易学易用

SQL 功能极强，完成核心功能只用了 9 个动词，如表 3-2 所示。SQL 语言接近英语口语，因此容易学习，容易使用。

表 3-2　SQL 语言的动词

SQL 功能	动　词
数据查询	SELECT
数据定义	CREATE，DROP，ALTER
数据操纵	INSERT，UPDATE，DELETE
数据控制	GRANT，REVOKE

3.1.3　SQL 的基本概念

数据模式是数据库系统中数据结构的一种表示形式，它具有不同的层次与结构方式，数据库系统的三级模式结构最早是在 1971 年由 DBTG 给出的，1975 年列入美国 ANSI/X3/SPARC 标准，它是一种数据库系统内部抽象结构体系。这三级模式结构分别介绍如下。

1. 概念模式（Conceptual Schema）

概念模式是数据库系统中全局数据逻辑结构的描述，是全体用户（应用）公共数据表，此种描述是一种抽象的描述，它不涉及具体的硬件环境与平台，也与具体的软件环境无关。

概念模式主要描述数据的概念记录类型、数据以及它们之间的关系，它还包括一些数据间的语义约束，对它的描述可用 DBMS 中的 DDL 语言定义。

【特别提示】可以从以下 4 个方面来理解概念模式:

① 一个表只有一个概念模式。

② 是数据库数据在逻辑级上的虚表。

③ 以某一种数据模型为基础。

④ 定义概念模式时不仅要定义数据的逻辑结构（如数据记录由哪些数据项构成，数据项的名字、类型、取值范围等），而且要定义与数据有关的安全性、完整性要求，定义这些数据之间的联系。

2. 用户模式（User's schema）

用户模式也称子模式（Subschema）或称外模式（External Schema），它是用户的数据视图，亦即是用户所见到的模式的一个部分，它由概念模式推导而出，概念模式给出了系统全局的数据描述，而外模式则给出每个用户的局部描述。一个概念模式可以有若干个外模式，每个用户只关心与它有关的模式，这样可以屏蔽大量无关信息且有利于数据保护，因此对用户极为有利。在一般的 DBMS 中都提供有相关的外模式描述语言（外模式 DDL）。

【特别提示】可以从以下 3 个方面来理解用户模式:

① 一个表可以有多个用户模式。

② 用户模式就是用户视图。

③ 用户模式是保证数据安全性的一个有力措施。

3. 物理模式（Physical Schema）

物理模式又称内模式（Internal Schema），它给出了数据库物理存储结构与物理存取方法，如数据存储的文件结构、索引、集簇及 hash 等存取方式与存取路径，内模式的物理性主要体现在操作系统及文件级上，它还不深入到设备级上（如磁盘及磁盘操作），但近年来有向设备级发展的趋势（如原始磁盘、磁盘分块技术等），DBMS 一般提供相关的内模式描述语言（内模式 DDL）。

【特别提示】可以从以下两个方面来理解物理模式:

① 一个数据库只有一个物理模式。

② 一个表可能由多个文件组成，如数据文件、索引文件。

数据模式给出了数据库的数据框架结构，而数据库中的数据才是真正的实体，但这些数据必须按框架所描述的结构组织，以概念模式为框架所组成的数据库叫做概念数据库（Conceptual Database），以用户模式为框架所组成的数据库叫做用户数据库（user's Database），以物理模式为框架所组成的数据库叫做物理数据库（Physical Database），这 3 种数据库中只有物理数据库是真实存在于计算机外存中，其他两种数据库并不真正存在于计算机中，而是通过两种映射由物理数据库映射而成。

模式的 3 个级别层次反映了模式的 3 个不同环境以及它们的不同要求，其中物理模式处于最低层，它反映了数据在计算机物理结构中的实际存储形式；概念模式处于中层，它反映了设计者的数据全局逻辑要求；而用户模式处于最外层，它反映了用户对数据的要求。

数据库系统的三级模式是对数据的 3 个级别抽象，它把数据的具体物理实现留给物理模式，使用户与全局设计者能不必关心数据库的具体实现与物理背景。同时，它通过两级映射建立三级模式间的联系与转换，使得概念模式与用户模式虽然并不具物理存在，但是也能通过映射而获得其存在的实体。同时两级映射也保证了数据库系统中数据的独立性，亦即数据的物理组织改变与逻辑概念级改变，并不影响用户模式的改变，它只要调整映射方式而不必改变用户模式。

（1）概念模式到物理模式的映射

该映射给出了概念模式中数据的全局逻辑结构到数据的物理存储结构间的对应关系，

此种映射一般由 DBMS 实现。

（2）外模式到概念模式的映射

概念模式是一个全局模式，而用户模式则是用户的局部模式，一个概念模式中可以定义多个用户模式，而每个用户模式是概念模式的一个基本视图。用户模式到概念模式的映射给出了用户模式与概念模式的对应关系，这种映射一般由 DBMS 实现。

SQL 语言支持关系数据库三级模式结构，如图 3-1 所示。

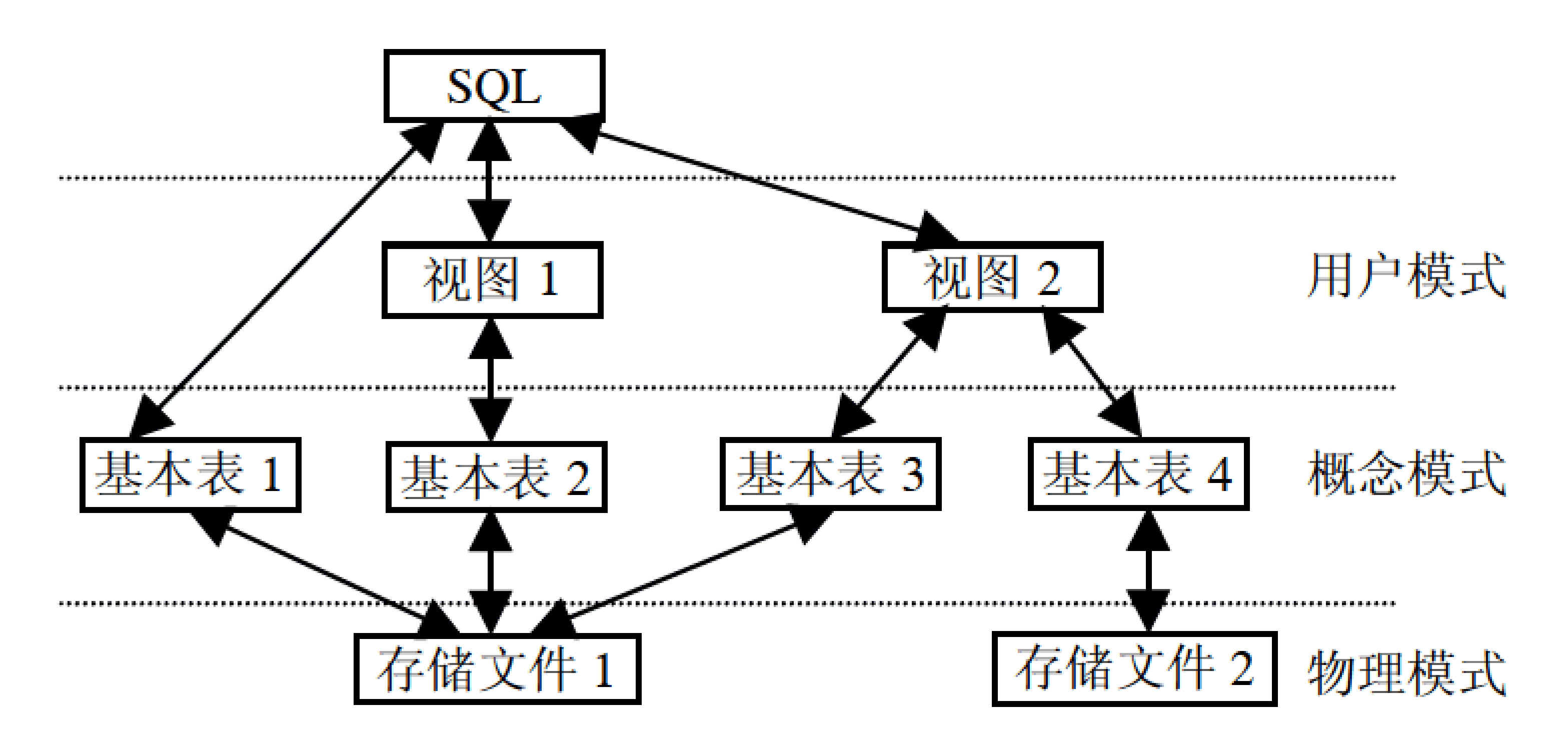


图 3-1　SQL 支持关系数据库三级模式结构

图中基本表是本身独立存在的表，在 SQL 中一个关系就对应一个基本表。一个（或多个）基本表对应一个存储文件，一个表可以带若干索引，索引也存放在存储文件中。

存储文件的逻辑结构组成了关系数据库的物理模式。存储文件的物理结构是任意的，对用户是透明的。

视图是从一个或几个基本表导出的表。数据库中只存放视图的定义而不存放视图对应的数据，这些数据仍存放在导出的视图的基本表中，因此视图是一个虚表。视图在概念上与基本表相同，用户可以在视图上再定义视图。

3.2　数据定义语言

数据库系统的三级模式中的基本对象有数据库、表、视图和索引。SQL 的数据定义功能包括定义数据库、定义表、定义视图和定义索引，如表 3-3 所示。

表 3-3　SQL 的数据定义语句

操作对象	操作方式		
	创　建	删　除	修　改
数据库	CREATE DATABASE	DROP DATABASE	
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	
索引	CREATE INDEX	DROP INDEX	

视图是基于基本表的虚表，索引是依附于基本表的，表、视图、索引等均为数据库组

成元素，因此 SQL 通常不提供数据库、视图和索引的直接修改操作，MySQL 支持视图的修改操作 ALTER VIEW。

【特别提示】用户如果想修改视图定义或索引定义，可以先将它们删除，然后再重建。

3.2.1　数据库级别的 SQL 操作

【特别提示】本章将使用 Navicat 8 for MySQL 软件（可到官方主页 http://www.navicat.com 下载）进行指导，使用其他图形界面管理工具（如 EMS SQL Management Studio 2007 for MySQL、MySQL-Front 和 sqlyog 等）操作大同小异。

启动 Navicat 8，单击“连线”按钮后，界面如图 3-2 所示。

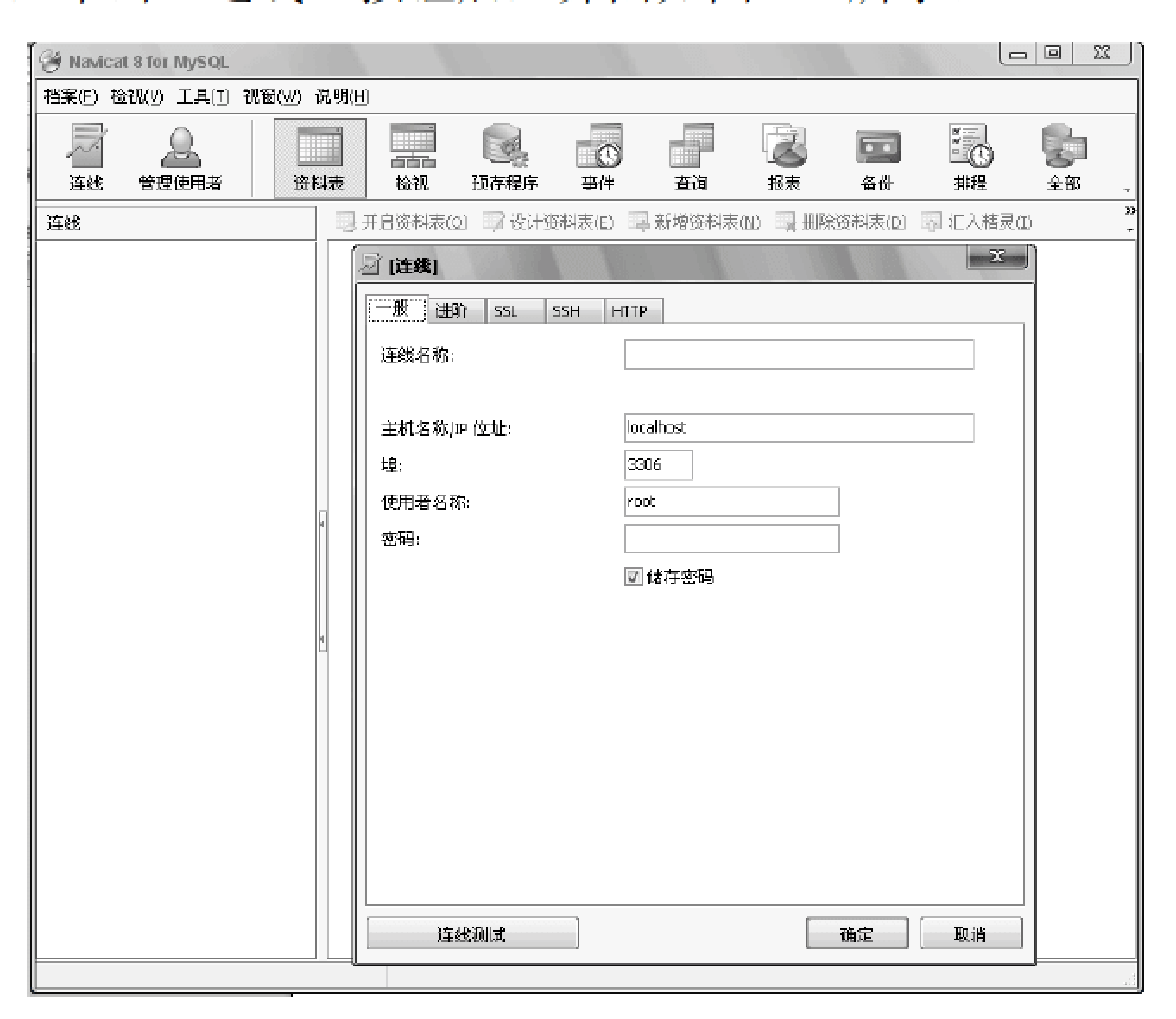


图 3-2　Navicat 8 for MySQL 界面

图中连接名称可自己定义，不填写默认为 localhost，如果 MySQL 端口不是默认的 3306，也需要更改为自己定义启动的端口，使用者名称默认为 root，密码默认为空或 root。

双击刚刚建立的连线（也称连接），然后按 F6 键启动“命令行控制台”。

1．用 Create Dabase 创建数据库

语法：CREATE DATABASE db_name

功能：CREATE DATABASE 用给定的名字创建一个数据库。

如果数据库已经存在，发生一个错误。

【特别提示】在 MySQL 中的数据库实现为包含对应数据库中表的文件的目录。因为数据库在初始创建时没有任何表，CREATE DATABASE 语句只是在 MySQL 数据目录下面创建一个目录。另外，对数据库操作中所有句法中的“DATABASE”

均可用“SCHEMA”代替，如“CREATE DATABASE db_name”等价于“CREATE SCHEMA db_name”。

例 3-1　建立一个 MIS 数据库。

在命令行“mysql>”提示符下输入：

```
mysql>CREATE DATABASE MIS;
```

【特别提示】在命令行“mysql>”提示符中所有的命令语句及符号均应是半角英文输入法下输入的字符，要注意“;”与“；”、“,”与“，”，“''”与“”以及半角空格与全角空格的区别，全角空格也是以一个汉字处理，占用两个字节，而半角空格是标准 ASCII 码，只占用一个字节，这是许多初学者非常易犯的错误。

如果成功建立会提示“Query OK, 1 row affected”，否则会返回一个类似“ERROR 1007 : Can't create database 'mis'; database exists”这样的错误。

【特别提示】命令行结尾应以分号“;”结束，一行写不下可换行继续写，命令行中不区别大小写（即大小写不敏感）。

为了使数据库各表中统一字符集，避免出现乱码，可以对刚刚建立的 MIS 数据库进行必要的处理：右击 MIS 数据库，在弹出的快捷菜单中选择“资料库内容”命令，然后在资料库内容中的“字元集”下拉列表框中选择 utf8 -- UTF-8 Unicode 选项，在“整理”下拉列表框中选择 utf8_general_ci 选项，如图 3-3 和图 3-4 所示。

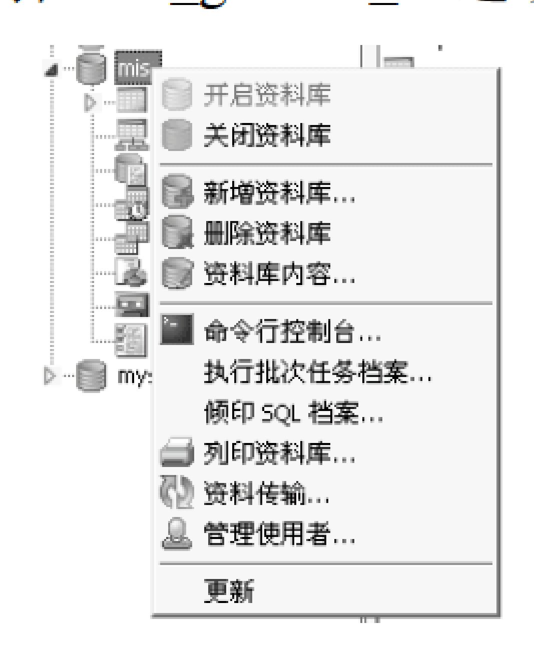


图 3-3　右击 mis 数据库时弹出的菜单



图 3-4　设置 mis 数据库的字符（元）集

2．用 SHOW 显示已有的数据库

语法：SHOW DATABASES [LIKE wild]

如果使用 LIKE wild 部分，wild 字符串可以是一个使用 SQL 的“%”和“_”通配符的字符串。

功能：SHOW DATABASES 列出在 MySQL 服务器主机上的数据库。

例 3-2　用 SHOW DATABASES 显示当前服务器主机上的数据库。

在命令行“mysql>”提示符下输入：

```
mysql>SHOW DATABASES;
```

显示结果如图 3-5 所示，此时在 Navcat 主窗口中按 F5 键刷新，可在连线窗口看到刚刚建立的 MIS 数据库。



图 3-5　命令行控制台运行 SHOW DATABASE 之后的界面

例 3-3　用 SHOW DATABASES 显示当前服务器主机上数据库名中第 3 个字母是“S”的所有数据库。

在命令行“mysql>”提示符下输入：

```
mysql>SHOW DATABASES LIKE "_s%";
```

返回结果如下：

```
+-----------------+
| Database (_s%) |
+-----------------+
| mis             |
| mysql           |
+-----------------+
2 rows in SET
```

3. 用 DROP DATABASE 删除数据库

语法：DROP DATABASE [IF EXISTS] db_name

功能：DROP DATABASE 删除数据库中的所有表和数据库。要慎重使用该命令。

DROP DATABASE 返回从数据库目录被删除的文件的数目。通常，这 3 倍于表的数量，因为每张表对应于一个“.MYD”文件、一个“.MYI”文件和一个“.FRM”文件。

在 MySQL 3.22 或以后版本中，可以使用关键词 IF EXISTS 阻止一个错误的发生，如果数据库不存在。

例 3-4　用 DROP DATABASE 删除刚刚建立的 MIS 数据库。

在命令行“mysql>”提示符下输入：

```
mysql>DROP DATABASE MIS;
```

4. 直接在数据库目录中创建或删除

用上述方法创建数据库，只是 MySQL 数据目录下面创建一个与数据库同名目录，同样删除数据库是把这个目录删除。

【特别提示】你可以直接这么做，创建、删除数据库或者给数据库更名，即对应相应的目录进行创建、删除和改名。用这种方法我们通常备份一个数据库，只需要把相应的文件夹复制到指定的地方，恢复时只需要把备份的文件夹复制到 mysql 的 data 目录下。

5. 用 USE 改变当前使用的数据库

语法：USE db_name

USE db_name 语句告诉 MySQL 使用 db_name 数据库作为随后的查询的当前数据库。数据库保持到会话结束，或直到发出另外一个 USE 语句。例如以下语句（如果读者无法理解 SELECT 语句，可以先跳过，在 3.3 节将介绍 SELECT 语句）。

```
mysql> USE mysql;
mysql> SELECT count(*) FROM mytable;   # SELECTs FROM mysql.mytable
mysql> USE mis;
mysql>SELECT count(*) FROM mytable; # SELECTs FROM mis.mytable
```

如果不是用 USE 语句，那么上面的例子应该写成：

```
mysql> SELECT count(*) FROM mysql.mytable;
mysql> SELECT count(*) FROM mis.mytable;
```

由于 use 也是一个 mysql 客户程序的命令，所以在命令行最后不加分号，客户程序可以得到结果。

3.2.2 创建、删除与修改基本表

1. 用 CREATE TABLE 语句创建数据表

用 CREATE TABLE 语句创建表。此语句的完整语法是相当复杂的，因为存在那么多的可选子句，但在实际中此语句的应用相当简单。其实，大多数复杂东西都是一些子句，我们暂时把这些子句剔除掉来学习基本的语法。

CREATE TABLE 语句的基本语法：

```
CREATE TABLE tbl_name(create_definition,...)
create_definition: col_name type [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [ UNIQUE ] [PRIMARY KEY]
```

在 MySQL 中，表名可以被指定为 db_name.tbl_name，不管有没有当前的数据库都可以。

例 3-5 在 MIS 数据库中创建一个访问者留言表。

在命令行“mysql>”提示符下输入：

```
mysql> CREATE DATABASE mis;
mysql>CREATE TABLE guestbook(visitor VARCHAR(40),comments TEXT,entrydate
DATETIME);
```

实际上更多时候这样写（结构比较清晰）：

```
mysql> CREATE  TABLE  guestbook
-> (
-> visitor VARCHAR(40),
-> comments TEXT,
-> entrydate DATETIME
->);
```

如果一切正常，则已经建立了第一个表。

所创建的表名为 guestbook，可以使用这个表来存储自站点访问者的信息。该表是由 CREATE TABLE 语句创建的，这个语句有两部分：第一部分指定表的名字；第二部分是括号中的各字段的名称和属性，相互之间用逗号隔开。

表 guestbook 有 3 个字段：visitor、comments 和 entrydate。visitor 字段存储访问者的名字，comments 字段存储访问者对站点的意见，entrydate 字段存储访问者访问站点的日期和时间。

注意每个字段名后面都跟有一个专门的表达式。例如，字段名 comments 后面跟有表达式 TEXT。这个表达式指定了字段的数据类型。数据类型决定了一个字段可以存储什么样的数据。因为字段 comments 包含文本信息，其数据类型定义为文本型。

【特别提示】在 mysql 中有如下 4 种数据类型。

（1）数值型

数值是诸如 268 或-478.2 这样的值。mysql 支持科学表示法，科学表示法由整数或浮点数后跟“E”或“e”、一个符号（“+”或“-”）和一个整数指数来表示。2.37E+13 和 12.69e-1 都是合法的科学表示法表示的数。而 7.37e41 不是合法的，因为指数前的符号未给出。

浮点数由整数部分、一个小数点和小数部分组成。整数部分和小数部分可以分别为空，但不能同时为空。

数值前可放一个负号“-”以表示负值。

（2）字符（串）型

字符型（也叫做字符串型，简称串）是诸如“I am a student”或“不能说的秘密”这样的值，或者是电话号码 88320109 这样的值。既可用单引号也可用双引号将串值括起来。

初学者往往分不清数值 88320109 和字符串 88320109 的区别。都是数字，为什么一个要用数值型，一个要用字符型呢？关键就在于：数值型的 888320109 是要参与计算的，例如它是一笔存款数；而字符型的 88320109 是不参与计算的，只是表示电话号码，这样的还有邮政编码、QQ 号码等，它们都不参与计算。

（3）日期和时间型

日期和时间是一些诸如“2007-09-12”或“11:23:51”这样的值。mysql 还支持日期/时间的组合，如“2007-09-12 11:23:51”。

（4）NULL 值

NULL 是一种“无类型”的值。它过去常表示的意思是“无值”、“未知值”、“丢失的值”、“溢出值”以及“没有上述值”等。可将 NULL 值插入表中、从表中检索它们，测试某个值是否是 NULL，但不能对 NULL 值进行算术运算（如果对 NULL 进行算术运算，其结果为 NULL）。例如填写表格中通信地址不清楚留空不填写，这就是 NULL 值。

用 create table 语句创建一个表，这个表中包含列的定义。例如在例 3-5 中创建了一个 guestbook 表，这个表中有 visitor 和 comments 两个列，如果要求这两列中 vistor 不允许空值，而 comments 允许空值，则例 3-5 可以改写为如下形式：

```
mysql> CREATE  TABLE  guestbook
-> (
-> visitor VARCHAR(40) NOT NULL,
-> comments TEXT NULL,
-> entrydate DATETIME
->);
```

定义一个列的语法如下：

```
col_name col_typy [col_attributes] [general_attributes]
```

其中列名由 col_name 给出。列名可最多包含 64 个字符，字符包括字母、数字、下划线及美元符号（$）。列名可以名字中合法的任何符号（包括数字）开头。但列名不能完全由数字组成，因为那样可能使其与数据分不开。mysql 保留诸如 SELECT、delete 和 create 这样的词，这些词不能用作列名，但函数名（如 pos 和 min）是可以使用的。

列类型 col_type 表示列可存储的特定值。列类型说明符还能表示存放在列中的值的最大长度。对于某些类型，可用一个数值明确地说明其长度。而另外一些值，其长度由类型名蕴含。例如，char(10) 明确指定了 10 个字符的长度，而 tinyblob 值隐含最大长度为 255 个字符。有的类型说明符允许指定最大的显示宽度（即显示值时使用多少个字符）。浮点类型允许指定小数位数，所以能控制浮点数的精度值为多少。

【特别提示】mysql 的列（字段）类型有如下 3 种。

数据库中的每个表都是由一个或多个列（字段）构成的。在用 create table 语句创建一个表时，要为每列（字段）指定一个类型。列（字段）的类型比数据类型更为细化，它精确地描述了给定表列（字段）可能包含的值的种类，如是否带小数、是否文字很多。

（1）数值列类型

mysql 有整数和浮点数值的列类型。整数列类型可以有符号也可以无符号。有一种特殊的属性允许整数列值自动生成，这对需要唯一序列或标识号的应用系统来说是非常有用的。每种数值类型的名称、说明、取值范围和占用字节数如表 3-4 和表 3-5 所示。

mysql 提供了 5 种整型：tinyint、smallint、mediumint、int 和 bigint。int 为 integer 的缩写。这些类型在可表示的取值范围上是不同的，整数列可定义为 unsigned 从而禁用负值，这使列的取值范围在 0 以上。各种类型的存储量需求也是不同的，取值范围较大的类型所需的存储量较大。

表 3-4　数值列类型说明

类　　型	说　　明
tinyint	非常小的整数
smallint	较小整数
mediumint	中等大小整数
int	标准整数
bigint	较大整数
float	单精度浮点数
double	双精度浮点数
decimal	一个串的浮点数

表 3-5　数值列类型取值范围及占用字节

类　　型	占 用 字 节	最　小　值	最　大　值
tinyint[(m)]	1	−128	127
unsigned tinyint[(m)]	1	0	255
smallint[(m)]	2	−32768	332767
unsigned smallint[(m)]	2	0	65535
mediumint[(m)]	3	−8388608	8388607
unsigned mediumint[(m)]	3	0	16777215
int[(m)]	4	−2147483648	2147483647
unsigned int[(m)]	4	0	4294967295
bigint [(m)]	8	−9223372036854775808	9223372036854775807
unsigned bigint [(m)]	8	0	18446744073709551615
float [(m,d)]	4	−1.175494351e+38	+1.175494351e+38
double[(m,d)]	8	−2.2250738585072014e+308	+2.2250738585072014e+308
decimal[(m,d)]	可变	其值的范围依赖于 m 和 d	

注：float 类型最小非零值：±1.175494351e−38

Double 类型最小非零值：±2.2250738585072014e−308

mysql 提供 3 种浮点类型：float、double 和 decimal。与整型不同，浮点类型不能是 unsigned 的，其取值范围也与整型不同，这种不同不仅在于这些类型有最大值，而且还有最小非零值。最小非零值提供了相应类型精度的一种度量，这对于记录科学数据来说是非常重要的（当然，也有负的最大和最小值）。

在选择了某种数值类型时，应该考虑所要表示的值的范围，只需选择能覆盖要取值的范围的最小类型即可。选择较大类型会对空间造成浪费，使表不必要地增大，处理起来没有选择较小类型那样有效。对于整型值，如果数据取值范围较小，如人员年龄或兄弟姐妹数，则 tinyint 最合适。mediumint 能够表示数百万的值并且可用于更多类型的值，但存储代价较大。bigint 在全部整型中取值范围最大，而且需要的存储空间是表示范围次大的整型 int 类型的两倍，因此只在确实需要时才用。对于浮点值，double 占用 float 的两倍空间。除非特别需要高精度或范围极大的值，一般应使用 float 型来表示数据。

在定义整型列时，可以指定可选的显示尺寸 m。如果这样，m 应该是一个 1~255 的整数。它表示用来显示列中值的字符数。例如，mediumint(4)指定了一个具有 4 个字符显示宽度的 mediumint 列。如果定义了一个没有明确宽度的整数列，将会自动分配给它一个默认的宽度。默认值为每种类型的“最长”值的长度。如果某个特定值的可打印表示需要不止 m 个字符，则显示完全的值；不会将值截断以适合 m 个字符。

对每种浮点类型，可指定一个最大的显示尺寸 m 和小数位数 d。m 的值应该取 1~255。d 的值可为 0~30，但不应大于 m−2。m 和 d 对 float 和 double 都是可选的，但对于 decimal 是必需的。如果省略了 m 和 d，则使用默认值。

【特别提示】 MySQL 还提供了 bit 数值类型，如 bit(M)，它占用的字节数大约为(M+7)/8 个字节。对于 FLOAT(p)这样的定义，如果 0 <= p <= 24 则占用 4 个字节，如果 25 <= p <= 53 则占用 8 个字节。另外，int 类型也可以写成 integer。

（2）字符串列类型

mysql 提供了几种存放字符数据的串类型，如表 3-6 所示。

表 3-6　字符串列类型说明

类　型　名	说　　明
char	定长字符串
varchar	可变长字符串
binary	定长字节字符串
varbinary	可变长字节字符串
tinyblob	非常小的 blob（二进制的对象）
blob	小 blob
mediumblob	中等的 blob
longblob	大 blob
tinytext	非常小的文本串
text	小文本串
mediumtext	中等文本串
longtext	大文本串
enum	枚举；列可赋予某个枚举成员
set	集合；列可赋予多个集合成员

表 3-7 给出了 mysql 定义串值列的类型，以及每种类型的最大尺寸和存储需求。对于可变长的列类型，各行的值所占的存储量是不同的，这取决于实际存放在列中的值的长度。这个长度在表中用 1 表示。1 以外所需的额外字节为存放该值的长度所需的字节数。mysql 通过存储值的内容及其长度来处理可变长度的值。这些额外的字节是无符号整数。注意，可变长类型的最大长度、此类型所需的额外字节数以及占用相同字节数的无符号整数之间的对应关系。例如，mediumblob 值可能最多 $2^{24}-1$ 字节长并需要 3 个字节记录其结果。3 个字节的整数类型 mediumint 的最大无符号值为 $2^{24}-1$，这并非偶然。

表 3-7　串列类型最大尺寸及存储需求

类 型 说 明	最 大 尺 寸	存 储 需 求
Char(m)	2^8−1 字节	m 字节
varchar(m)	2^{16}−1 字节	1+1 字节，其中|<=m
binary(m)	2^8−1 字节	m 字节
varbinary(m)	2^8−1 字节	1+1 字节，其中|<=m
Tinyblob,tinytext	2^8−1 字节	1+1 字节
Blob,text	2^{16}−1 字节	1+2 字节
Mediumblob,mediumtext	2^{24}−1 字节	1+3 字节
Longblob,longtext	2^{32}−1 字节	1+4 字节
enum(“valuel”,“value2”,…)	65535 个成员	1 或 2 字节
set(“valuel”,“value2”,…)	64 个成员	1、2、3、4 或 8 字节

（3）日期时间列类型

mysql 提供了几种时间值的列类型，它们分别是：date、datetime、time、timestamp 和 year。表 3-8 给出了 mysql 为定义存储日期和时间值所提供的这些类型，表 3-9 给出了每种类型的合法取值范围和存储需求。

表 3-8　日期时间列类型

类　型　名	说　　明
date	“yyyy-mm-dd”格式表示的日期值
time	“hh:mm:ss”格式表示的时间值
datetime	“yyyy-mm-dd hh:mm:ss”格式
timestamp	“yyyymmddhhmmss”格式表示的时间戳值
year	“yyyy”格式的年份值

表 3-9　日前时间列类型的取值范围和存储需求

类　型　名	取 值 范 围	存 储 需 求
date	“1000-01-01”～“9999-12-31”	3 字节
time	“-838:59:59”～“839:59:59”	3 字节
datetime	“1000-01-01 00:00:00”～“9999-12-31 23:59:59”	8 字节
timestamp	19700101000000 到 2037 年的某个时刻	4 字节
year	1901 到 2155	1 字节

建表时可以在列类型之后指定可选的类型说明属性，以及指定更多的常见属性。属性起修饰类型的作用，并更改其处理列值的方式，属性有以下类型：

- ❑ 专用属性用于指定列。例如，unsigned 属性只针对整型，而 binary 属性只用于 char 和 varchar。
- ❑ 通用属性除少数列之外可用于任意列。可以指定 NULL 或 NOT NULL 以表示某个列是否能够存放 NULL。还可以用 default、def_value 来表示在创建一个新行但未明确给出该列的值时，该列可赋予值 def_value。def_value 必须为一个常量；它不

能是表达式，也不能引用其他列。不能对 blob 或 text 列指定默认值。

如果想给出多个列的专用属性，可按任意顺序指定它们，只要它们跟在列类型之后、通用属性之前即可。类似地，如果需要给出多个通用属性，也可按任意顺序给出它们，只要将它们放在列类型和可能给出的列专用属性之后即可。

下面在 MIS 数据库中创建学生表 Student、课程表 Course 和学生选课表 SC 3 个表。

例 3-6　在 MIS 数据库中创建学生表 Student：

```
mysql>CREATE TABLE Student
          (Sno     CHAR(9) PRIMARY KEY,  /* 列级完整性约束条件*/
          Sname    CHAR(20) UNIQUE,      /* Sname 取唯一值*/
          Ssex     CHAR(2),
          Sage     SMALLINT,
          Sdept    CHAR(20)
         );
```

例 3-7　在 MIS 数据库中创建课程表 Course：

```
mysql>CREATE TABLE  Course
           ( Cno     CHAR(4) PRIMARY KEY,
            Cname    CHAR(40),
            Cpno     CHAR(4) ,
            Ccredit   SMALLINT,
            FOREIGN KEY (Cpno) REFERENCES  Course(Cno)
        );
```

例 3-8　在 MIS 数据库中创建学生选课表 SC：

```
mysql>CREATE TABLE  SC
        (Sno  CHAR(9),
        Cno   CHAR(4),
        Grade    SMALLINT,
        PRIMARY KEY (Sno,Cno),
                  /* 主码由两个属性构成，必须作为表级完整性进行定义*/
        FOREIGN KEY (Sno) REFERENCES Student(Sno),
                  /* 表级完整性约束条件，Sno 是外码，被参照表是 Student */
        FOREIGN KEY (Cno) REFERENCES Course(Cno)
                  /* 表级完整性约束条件， Cno 是外码，被参照表是 Course*/
        );
mysql> describe Student;
```

建好表之后，可以用 SHOW COLUMNS 来显示给定表中的各列的信息。

SHOW COLUMNS 语句的基本语法：

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
```

例 3-9　显示表 SC 中各列信息：

```
mysql> show columns FROM sc;
```

返回结果如下：

```
+--------+--------------+------+-----+---------+--------+
| Field  | Type         | Null | Key | Default | Extra  |
+--------+--------------+------+-----+---------+--------+
| Sno    | char(9)      | NO   | PRI |         |        |
| Cno    | char(4)      | NO   | PRI |         |        |
| Grade  | smallint(6)  | YES  |     | NULL    |        |
+--------+--------------+------+-----+---------+--------+
3 rows in set
```

也可以用 SHOW FIELDS 代替 SHOW COLUMNS 语句。

例 3-10　显示表 Course 中各列信息：

```
mysql> show fields FROM Course;
```

返回结果如下：

```
+---------+--------------+------+-----+---------+--------+
| Field   | Type         | Null | Key | Default | Extra  |
+---------+--------------+------+-----+---------+--------+
| Cno     | char(4)      | NO   | PRI |         |        |
| Cname   | char(40)     | YES  |     | NULL    |        |
| Cpno    | char(4)      | YES  | MUL | NULL    |        |
| Ccredit | smallint(6)  | YES  |     | NULL    |        |
+---------+--------------+------+-----+---------+--------+
4 rows in set
```

DESCRIBE 可以提供有关表中各列的信息。它是 SHOW COLUMNS FROM 的快捷方式。

例 3-11　显示表 Student 中各列信息：

```
mysql> describe Student;
```

返回结果如下：

```
+--------+--------------+------+-----+---------+--------+
| Field  | Type         | Null | Key | Default | Extra  |
+--------+--------------+------+-----+---------+--------+
| Sno    | char(9)      | NO   | PRI |         |        |
| Sname  | char(20)     | YES  | UNI | NULL    |        |
| Ssex   | char(2)      | YES  |     | NULL    |        |
| Sage   | smallint(6)  | YES  |     | NULL    |        |
| Sdept  | char(20)     | YES  |     | NULL    |        |
+--------+--------------+------+-----+---------+--------+
5 rows in set
```

2．用 ALTER TABLE 语句修改表的结构

有时可能需要改变一下现有表的结构，那么 Alter Table 语句将是合适的选择。

增加列：alter table tbl_name add col_name type

例 3-12　给表 guestbook 增加一列 IPaddr：

```
mysql>alter table guestbook add IPaddr char(15);
```

删除列：alter table tbl_name drop col_name

例 3-13　删除列 IPaddr：

```
mysql>alter table guestbook drop IPaddr;
```

改变列：alter table tbl_name modify col_name type

例 3-14　改变列 IPaddr 的类型：

```
mysql> alter table guestbook modify IPaddr int;
```

另一种方法是：alter table tbl_name change old_col_name col_name type

例 3-15　改变列 IPaddr 的类型：

```
mysql> alter table guestbook change IPaddr IPaddr char(15);
```

例 3-16　给列 IPaddr 更名为 IPaddress：

```
mysql>alter table guestbook change IPaddr IPaddress;
```

给表更名：alter table tbl_name rename new_tbl

例 3-17　把 guestbook 表更名为 mybook：

```
mysql>alter table guestbook rename mybook;
```

3. 用 DROP TABLE 语句删除数据表

DROP TABLE [IF EXISTS] tbl_name [, tbl_name,...]

DROP TABLE 删除一个或多个数据库表。所有表中的数据和表定义均被删除，故慎重使用这个命令。

在 MySQL 中可以使用关键词 IF EXISTS 类避免不存在表的一个错误发生。

例 3-18　把 mis 数据库中的 mybook 删除：

```
mysql>USE mis;
mysql>DROP TABLE guestbook;
```

或者，也可以同时指定数据库和表：

```
mysql>DROP TABLE mis. guestbook;
```

3.3　数据查询语言

数据库查询是数据库的核心操作。SQL 语言提供了 SELECT 语句进行数据库的查询，该语句具有灵活的使用方式和丰富的功能。本节介绍 SELECT 语句关于查询的最基本功能。

SELECT 语句的语法如下：

- ❑ SELECT column_list（选择哪些列）。
- ❑ FROM table_list（从何处选择行）。
- ❑ WHERE primary_constraint（行必须满足什么条件）。
- ❑ GROUP BY grouping_columns（怎样对结果分组）。
- ❑ HAVING secondary_constraint（行必须满足的第二条件）。

- ORDER BY sorting_columns（怎样对结果排序）。
- LIMIT count（结果限定）。

注意：所有使用的关键词必须严格按上面的顺序给出。例如，一个 HAVING 子句必须跟在 GROUP BY 子句之后和 ORDER BY 子句之前。

【特别提示】除了 SELECT 和 column_list 部分外，语法中的每个选项都是可选的，有的数据库还需要 FROM 子句。MySQL 有所不同，它允许对表达式求值而不引用任何表。

例 3-19　用 SELECT 计算 3.1415926*2、3.1415926E+3+2、–3.1415926E–3+2 的值：

```
mysql> SELECT 3.1415926*2,3.1415926E+3+2, -3.1415926E-3+2;
```

返回结果如下：

```
+-------------+----------------+-----------------+
| 3.1415926*2 | 3.1415926E+3+2 | -3.1415926E-3+2 |
+-------------+----------------+-----------------+
| 6.2831852   |      3143.5926 |    1.9968584074 |
+-------------+----------------+-----------------+
1 row in set
```

3.3.1　普通查询

SELECT 最简单的形式是从一张表中检索每样东西。

例 3-20　查询 Student 表中所有学生记录。

```
mysql> SELECT * FROM Student;
```

返回结果如下：

```
+-----------+--------+------+------+--------+
| Sno       | Sname  | Ssex | Sage | Sdept  |
+-----------+--------+------+------+--------+
| 072101001 | 张三   | 男   |   19 | 信息系 |
| 072101002 | 李四   | 女   |   20 | 信息系 |
| 072101003 | 王五   | 男   |   20 | 信息系 |
| 072102001 | 李晨浩 | 男   |   21 | 物理系 |
| 072102002 | 刘敏菲 | 女   |   19 | 物理系 |
| 072102003 | 张勇云 | 男   |   20 | 物理系 |
| 072101004 | 方月红 | 女   |   20 | 信息系 |
+-----------+--------+------+------+--------+
7 rows in set
```

SELECT 也可查询特定行。

例 3-21　查询学生“方月红”的相关信息。

```
mysql> SELECT *
       FROM Student
       WHERE Sname = "方月红";
```

返回结果如下：

```
+-----------+--------+------+------+--------+
| Sno       | Sname  | Ssex | Sage | Sdept  |
+-----------+--------+------+------+--------+
| 072101004 | 方月红 | 女   |   20 | 信息系 |
+-----------+--------+------+------+--------+
1 row in set
```

可以对照例 3-20 来验证查询的结果是否一致。

SELECT 也可查询特定列。

例 3-22　查询 Student 表中所有信息系学生姓名和年龄。

```
mysql> SELECT Sname, Sage
       FROM Student
       WHERE Sdept="信息系";
```

返回结果如下：

```
+--------+------+
| Sname  | Sage |
+--------+------+
| 张三   |   19 |
| 李四   |   20 |
| 王五   |   20 |
| 方月红 |   20 |
+--------+------+
4 rows in set
```

SELECT 还可进行表达式计算。前面的多数查询通过从表中检索值已经产生了输出结果。MySQL 还允许作为一个公式的结果来计算输出列的值。表达式可以简单也可以复杂。

例 3-23　计算表 Student 中学生“方月红”的出生年份。

```
mysql> SELECT 2007-Sage
       FROM Student
       WHERE Sname="方月红 ";
```

返回结果如下：

```
+-----------+
| 2007-Sage |
+-----------+
|      1987 |
+-----------+
1 row in set
```

3.3.2　条件查询

不必每次查询都返回所有的行记录，可以使用 WHERE 或者 HAVING 从句。HAVING 从句与 WHERE 从句的区别是，HAVING 表达的是第二条件，再与其他从句配合使用，显然不能在 WHERE 子句中的项目使用 HAVING。因此本小节仅介绍 WHERE 从句的使用，HAVING 从句的使用方法类似。另外，WHERE 从句也可以实现 HAVING 从句的绝大部分功能。

为了限制 SELECT 语句检索出来的记录集，可使用 WHERE 子句，它给出选择行的条件。可通过查找满足各种条件的列值来选择行。

WHERE 子句中的表达式可使用表 3-10 中的算术运算符、表 3-11 中的比较运算符和表 3-12 中的逻辑运算符。还可以使用圆括号将一个表达式分成几个部分。可使用常量、表列和函数来完成运算。在本教程的查询中，有时使用几个 MySQL 函数，但是 MySQL 的函数远不止这里所给出的。

表 3-10　算术运算符

运　算　符	说　　明	运　算　符	说　　明
+	加	*	乘
-	减	/	除

表 3-11　比较运算符

运　算　符	说　　明	运　算　符	说　　明
<	小于	!=或<>	不等于
<=	小于或等于	>=	大于或等于
=	等于	>	大于

表 3-12　逻辑运算符

运　算　符	说　　明
NOT 或!	逻辑非
OR 或||	逻辑或
AND 或&&	逻辑与

例 3-21～例 3-23 已经在使用 WHERE 子句进行条件查询。当然，可以在任何列上指定条件或者进行条件组合查询。

例 3-24　查询 Student 表中所有 1987 年后出生的学生信息及对应的出生年份。

```
mysql> SELECT *, 2007-Sage
       FROM Student
       WHERE 2007-Sage>=1987;
```

返回结果如下：

```
+-----------+--------+------+------+--------+-----------+
| Sno       | Sname  | Ssex | Sage | Sdept  | 2007-Sage |
+-----------+--------+------+------+--------+-----------+
| 072101001 | 张三   | 男   |   19 | 信息系 |      1988 |
| 072101002 | 李四   | 女   |   20 | 信息系 |      1987 |
| 072101003 | 王五   | 男   |   20 | 信息系 |      1987 |
| 072102002 | 刘敏菲 | 女   |   19 | 物理系 |      1988 |
| 072102003 | 张勇云 | 男   |   20 | 物理系 |      1987 |
| 072101004 | 方月红 | 女   |   20 | 信息系 |      1987 |
+-----------+--------+------+------+--------+-----------+
6 rows in set
```

采用 AND 逻辑运算符的查询。

例 3-25　查询 Student 表中所有 1987 年后出生的信息系学生信息及对应的出生年份。

```
mysql> SELECT *, 2007-Sage
       FROM Student
       WHERE 2007-Sage>=1987 AND Sdept="信息系";
```

返回结果如下：

```
+-----------+--------+------+------+--------+-----------+
| Sno       | Sname  | Ssex | Sage | Sdept  | 2007-Sage |
+-----------+--------+------+------+--------+-----------+
| 072101001 | 张三   | 男   |   19 | 信息系 |      1988 |
| 072101002 | 李四   | 女   |   20 | 信息系 |      1987 |
| 072101003 | 王五   | 男   |   20 | 信息系 |      1987 |
| 072101004 | 方月红 | 女   |   20 | 信息系 |      1987 |
+-----------+--------+------+------+--------+-----------+
4 rows in set
```

采用 OR 逻辑运算符的查询。

例 3-26　查询 Student 表中所有 1988 年后或者 1986 年以前出生的学生信息及对应的出生年份。

```
mysql>SELECT*,2007-Sage FROM Student WHERE 2007-Sage>=1988 or 2007-Sage
<=1986;
```

返回结果如下：

```
+-----------+--------+------+------+--------+-----------+
| Sno       | Sname  | Ssex | Sage | Sdept  | 2007-Sage |
+-----------+--------+------+------+--------+-----------+
| 072101001 | 张三   | 男   |   19 | 信息系 |      1988 |
| 072102001 | 李晨浩 | 男   |   21 | 物理系 |      1986 |
| 072102002 | 刘敏菲 | 女   |   19 | 物理系 |      1988 |
+-----------+--------+------+------+--------+-----------+
3 rows in set
```

【特别提示】例 3-26 也可采用 NOT 逻辑运算符的查询。

```
mysql>  SELECT *, 2007-Sage
        FROM Student
        WHERE NOT (2007-Sage=1987);
```

还可以写成：

```
mysql>  SELECT *, 2007-Sage
        FROM Student
        WHERE 2007-Sage<>1987;
```

返回的结果均与例 3-26 一致。

例 3-27　查询 Student 表中所有 1988 年后或者 1986 年以前出生的信息系学生信息及对应的出生年份。

```
mysql>SELECT*,2007-Sage FROM Student WHERE(2007-Sage>=1988 or 2007-Sage<
=1986) AND Sdept="信息系";
```

返回结果如下：

```
+-----------+-------+------+------+--------+-----------+
| Sno       | Sname | Ssex | Sage | Sdept  | 2007-Sage |
+-----------+-------+------+------+--------+-----------+
| 072101001 | 张三  | 男   |   19 | 信息系 |      1988 |
+-----------+-------+------+------+--------+-----------+
1 row in set
```

【特别提示】当 OR 与 AND 连用时，一定要注意运算的优先次序，由于 AND 比 OR 优先运算，如果要改变它们的运算次序，可以用小括号。例 3-27 也可做进一步简化。

```
mysql>SELECT*, 2007-Sage FROM Student WHERE 2007-Sage<>1987 AND Sdept="信
息系";
```

3.3.3　查询排序

使用 ORDER BY 子句对查询返回的结果按一列或多列排序。ORDER BY 子句的语法格式为：

ORDER BY column_name [ASC|DESC] [,…]

其中 ASC 表示升序，为默认值，DESC 为降序。ORDER BY 不能按 ntext、text 和 image 数据类型进行排序。另外，可以根据表达式进行排序。

例 3-28　表 Student 中所有学生按年龄排序。

```
mysql> SELECT *
       FROM Student
       ORDER BY Sage;
```

返回结果如下：

```
+-----------+--------+------+------+--------+
| Sno       | Sname  | Ssex | Sage | Sdept  |
+-----------+--------+------+------+--------+
| 072101001 | 张三   | 男   |   19 | 信息系 |
| 072102002 | 刘敏菲 | 女   |   19 | 物理系 |
| 072101002 | 李四   | 女   |   20 | 信息系 |
| 072101003 | 王五   | 男   |   20 | 信息系 |
| 072102003 | 张勇云 | 男   |   20 | 物理系 |
| 072101004 | 方月红 | 女   |   20 | 信息系 |
| 072102001 | 李晨浩 | 男   |   21 | 物理系 |
+-----------+--------+------+------+--------+
7 rows in set
```

如果以逆序排序，增加 DESC（下降）关键字到需要排序的列名上。

例 3-29　表 Student 中所有学生按年龄逆序排序。

```
mysql> SELECT *
       FROM Student
       ORDER BY Sage DESC;
```

返回结果如下：

```
+-----------+--------+------+------+--------+
| Sno       | Sname  | Ssex | Sage | Sdept  |
+-----------+--------+------+------+--------+
| 072102001 | 李晨浩 | 男   |   21 | 物理系 |
| 072101002 | 李四   | 女   |   20 | 信息系 |
| 072101003 | 王五   | 男   |   20 | 信息系 |
| 072102003 | 张勇云 | 男   |   20 | 物理系 |
| 072101004 | 方月红 | 女   |   20 | 信息系 |
| 072101001 | 张三   | 男   |   19 | 信息系 |
| 072102002 | 刘敏菲 | 女   |   19 | 物理系 |
+-----------+--------+------+------+--------+
7 rows in set
```

也可以在多个列上排序。

例 3-30　表 Student 中所有学生按系别、年龄（逆序）、性别分别排序。

```
mysql> SELECT *
       FROM Student
       ORDER BY Sdept,Sage DESC,Ssex;
```

返回结果如下：

```
+-----------+--------+------+------+--------+
| Sno       | Sname  | Ssex | Sage | Sdept  |
+-----------+--------+------+------+--------+
| 072101002 | 李四   | 女   |   20 | 信息系 |
| 072101004 | 方月红 | 女   |   20 | 信息系 |
| 072101003 | 王五   | 男   |   20 | 信息系 |
| 072101001 | 张三   | 男   |   19 | 信息系 |
| 072102001 | 李晨浩 | 男   |   21 | 物理系 |
| 072102003 | 张勇云 | 男   |   20 | 物理系 |
| 072102002 | 刘敏菲 | 女   |   19 | 物理系 |
+-----------+--------+------+------+--------+
7 rows in set
```

3.3.4　查询分组与行计数

GROUP BY 从句根据所给的列名返回分组的查询结果，可用于查询具有相同值的列。其语法为：

GROUP BY col_name,…

例 3-31　表 Student 中所有学生按系别分组。

```
mysql>  SELECT *
        FROM Student
        GROUP BY Sdept;
```

返回结果如下：

```
+-----------+--------+------+------+--------+
| Sno       | Sname  | Ssex | Sage | Sdept  |
+-----------+--------+------+------+--------+
| 072101001 | 张三   | 男   |   19 | 信息系 |
| 072102001 | 李晨浩 | 男   |   21 | 物理系 |
+-----------+--------+------+------+--------+
2 rows in set
```

由以上结果可以看出，查询显示结果时，被分组的列如果有重复的值，只返回靠前的记录，并且返回的记录集是排序的。仅使用 GROUP BY 从句并没有什么意义，该从句的真正作用在于与各种组合函数配合，用于行计数。现在利用 COUNT()函数计数非 NULL 结果的数目。

例 3-32　计算 Student 表中的记录数。

```
mysql> SELECT count(*) FROM Student;
```

返回结果如下：

```
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set
```

例 3-33　计算 Student 表中性别非空的记录数。

```
mysql> SELECT count(Ssex) FROM Student;
```

返回结果如下：

```
+-------------+
| count(Ssex) |
+-------------+
|           7 |
+-------------+
1 row in set
```

现在配合 GROUP BY 从句使用。

例 3-34　计算 Student 表中各系别的学生数。

```
mysql>  SELECT Sdept,count(*)
```

```
        FROM Student
        GROUP BY Sdept;
```

返回结果如下：

```
+---------+-----------+
| Sdept   | count(*)  |
+---------+-----------+
| 信息系  |         4 |
| 物理系  |         3 |
+---------+-----------+
2 rows in set
```

例 3-35　计算 Student 表中男女学生数。

```
mysql>  SELECT Ssex,count(*)
        FROM Student
        GROUP BY Ssex;
```

返回结果如下：

```
+------+-----------+
| Ssex | count(*)  |
+------+-----------+
| 女   |         3 |
| 男   |         4 |
+------+-----------+
2 rows in set
```

【特别提示】除了计数还需要返回一个列的值，那么必须使用 GROUP BY 语句，否则无法计算记录。也可以根据多个列分组。

例 3-36　计算 Student 表中各系别的男女学生数。

```
mysql>  SELECT Sdept,Ssex,count(*)
        FROM Student
        GROUP BY Sdept,Ssex;
```

返回结果如下：

```
+---------+------+-----------+
| Sdept   | Ssex | count(*)  |
+---------+------+-----------+
| 信息系  | 女   |         2 |
| 信息系  | 男   |         2 |
| 物理系  | 女   |         1 |
| 物理系  | 男   |         2 |
+---------+------+-----------+
4 rows in set
```

HAVING 子句是用来对 GROUP BY 所分组的记录进行筛选，使用时必须跟在 GROUP BY 子句之后。

例 3-37　选出 Student 表中各系别的男女学生数大于 1 的记录。

```
mysql>  SELECT Sdept,Ssex,count(*)
        FROM Student
        GROUP BY Sdept,Ssex
        HAVING count(*)>1;
```

返回结果如下：

```
+--------+------+----------+
| Sdept  | Ssex | count(*) |
+--------+------+----------+
| 信息系 | 女   |        2 |
| 信息系 | 男   |        2 |
| 物理系 | 男   |        2 |
+--------+------+----------+
3 rows in set
```

例 3-38　选出 Student 表中各系别的男女学生数大于 1 的详细情况，并按性别排序。

```
mysql>  SELECT Sdept,Ssex,count(*)
        FROM Student
        GROUP BY Sdept,Ssex
        HAVING count(*)>1
        ORDER BY Ssex;
```

返回结果如下：

```
+--------+------+----------+
| Sdept  | Ssex | count(*) |
+--------+------+----------+
| 信息系 | 女   |        2 |
| 物理系 | 男   |        2 |
| 信息系 | 男   |        2 |
+--------+------+----------+
3 rows in set
```

3.3.5　多表查询

所谓多表查询是相对单表而言的，指从多个数据表中查询数据，也称为连接查询。连接查询是关系数据库中最主要的查询，包括等值连接、自然连接、非等值连接查询、自身连接查询和复合条件连接查询等。

多表查询中用来连接两个表的条件称为连接条件，格式为：

[<表名 1>].<列名 1> <比较运算符> [<表名 2>].<列名 1>

其中比较运算符主要有：=、>、<、>=、<=、!=。连接的条件还可以使用下面形式：

[<表名 1>].<列名 1> BETWEEN [<表名 2>].<列名 1> AND [<表名 2>].<列名 2>

【特别提示】 当连接运算符为“=”时，称为等值连接，使用其他运算符称为非等值连接。执行连接的过程是：首先在表 1 中找到第一个记录，然后从头开始扫描表 2，逐一查找满足条件的记录，找到后就将表 1 中的第一个记录与该记录拼接起来，形成结果表中一个元组。表 2 全部查找完后，再找表 1 中第 2 个记录，然后再从头开始扫描表 2，逐一查找满足连接条件的记录，找到后就将表 1 中的第 2 个记录与该记录拼接起来，形成结果表中的一条记录。重复上述过程，直到表 1 中的全部记录都处理完毕为止。

当查询来自多个表的信息时，需要指定在一个表中的记录怎样能匹配其他表的记录，可使用 WHERE 子句基于列名值来匹配两个表中的记录。当引用两个表相同列名时，一定要指定哪个表，可把表名附在列名前。

例 3-39　查询每个学生及其选修课的情况。

```
mysql> SELECT Student.*,sc.*
       FROM Student,sc
       WHERE Student.Sno=sc.Sno;
```

返回结果如下：

Sno	Sname	Ssex	Sage	Sdept	Sno	Cno	Grade
072101001	张三	男	19	信息系	072101001	0001	92
072101001	张三	男	19	信息系	072101001	0002	85
072101001	张三	男	19	信息系	072101001	0003	88
072101002	李四	女	20	信息系	072101002	0002	90
072101002	李四	女	20	信息系	072101002	0003	80

5 rows in set

【特别提示】本例中，SELECT 子句与 WHERE 子句中的属性名前都加上了表名前缀，这是为了避免混淆。如果属性名在参加连接的各表中是唯一的，则可以省略表名前缀。

若在等值连接中把目标列中重复的属性去掉则为自然连接。

例 3-40　对例 3-39 用自然连接完成。

```
mysql> SELECT Student.Sno,Sname,Ssex,Sage,Sdept,Cno,grade
       FROM Student,SC
       WHERE Student.Sno=SC.Sno;
```

返回结果如下：

sno	sname	ssex	sage	sdept	cno	grade
072101001	张三	男	19	信息系	0001	92
072101001	张三	男	19	信息系	0002	85
072101001	张三	男	19	信息系	0003	88
072101002	李四	女	20	信息系	0002	90
072101002	李四	女	20	信息系	0003	80

5 rows in set

例中，由于 Sname、Ssex、Sage、Sdept、Cno 和 grade 属性列在 Student 表与 SC 表中是唯一的，因此引用时可以去掉表名前缀。而 Sno 两个表都出现了，因此引用时必须加上表名前缀。

连接操作不仅可以在两个表之间进行，也可以是一个表与其自己进行连接，称为表的自身连接。

例 3-41　查询每一门课的间接先修课程（即先修课程的先修课程）。

在 course 表关系中，只有每门课的直接先修课程信息，而没有先修课程的先修课。要得到这个信息，必须先对一门课找到其先修课，再按此先修课程的课程号，查找它的先修课程。这就要将 course 表与其自身连接。

为清楚起见，可以为 course 表取两个别名，一个是 firstCourse，另一个是 secondCourse。假设 course 中目前有以下记录值：

```
+------+------------------+------+---------+
| Cno  | Cname            | Cpno | Ccredit |
+------+------------------+------+---------+
| 0005 | 复变函数         | 0002 |       2 |
| 0001 | 大学英语         | 1001 |      12 |
| 0002 | 高等数学         | 1002 |       6 |
| 0003 | 马克思主义哲学   | 1003 |       2 |
| 0004 | 邓小平理论概论   | 0003 |       2 |
| 0006 | 电工学           | 0005 |       4 |
+------+------------------+------+---------+
6 rows in set
```

完成该查询的 SQL 语句为：

```
mysql> SELECT a.cno, b.cpno
       FROM course a,course b
       WHERE a.cpno=b.cno;
```

返回结果如下：

```
+------+------+
| cno  | cpno |
+------+------+
| 0005 | 1002 |
| 0004 | 1003 |
| 0006 | 0002 |
+------+------+
3 rows in set
```

连接操作除了可以是两表连接，一个表与其自身的连接外，还可以是两个表以上进行连接，即多表连接。

例 3-42　查询每个学生的学名、姓名、选修的课程名及成绩。

```
mysql> SELECT Student.Sno,Sname,Cname,grade
       FROM Student,sc,course
       WHERE Student.Sno=sc.Sno and sc.cno=course.cno;
```

返回结果如下：

```
+-----------+-------+----------------+-------+
| sno       | sname | cname          | grade |
+-----------+-------+----------------+-------+
| 072101001 | 张三  | 大学英语       |    92 |
| 072101001 | 张三  | 高等数学       |    85 |
| 072101001 | 张三  | 马克思主义哲学 |    88 |
| 072101002 | 李四  | 高等数学       |    90 |
| 072101002 | 李四  | 马克思主义哲学 |    80 |
+-----------+-------+----------------+-------+
5 rows in set
```

3.4　数据操纵语言

SQL 中数据操纵主要包括插入数据、修改数据和删除数据。

3.4.1　插入数据

1. 使用 INSERT 语句插入新数据

语法：INSERT [INTO] tbl_name [(col_name,...)] VALUES (pression,...),…

INSERT [INTO] tbl_name SET col_name=expression, ...

下面利用 INSERT 语句来增加记录，这是一个 SQL 语句，需要为它指定希望插入数据行的表或将值按行放入的表。INSERT 语句具有几种形式，可指定所有列的值。

例 3-43　mysql> INSERT INTO Student VALUES ('072101001', '张三', '男', 19, '信息系');

例 3-44　mysql>INSERT INTO Student(Sno, Sname , Ssex , Sage , Sdept) VALUES ('072101002', '李四', '女', 20, '信息系');

例 3-45　mysql>INSERT INTO Student SET Sno='072101003', Sname='王五', Ssex='男', Sage= 20, Sdept= '信息系';

例 3-46　使用多个值表，可以一次提供多行数据：

```
mysql>INSERT Student VALUES ('072102001', '李晨浩', '男', '21', '物理系'),
('072102002', '刘敏菲', '女', '19', '物理系'), ('072102003', '张勇云', '男',
'20','物理系');
```

【特别提示】“INTO” 一词是可选的。VALUES 后必须包含表中每列的值，并且按表中列的存放次序给出（这就是创建表时列的定义次序。如果不能肯定，可使用 DESCRIBE tbl_name 来查看这个次序）。

还可以给出要赋值的那个列，然后再列出值。这对于希望建立只有几个列需要初始设置的记录是很有用的。

例 3-47　mysql>INSERT INTO Student(Sno) VALUES('072101004');

这种形式的 INSERT 也允许多个值表。

例 3-48　mysql>INSERT INTO Student (Sno) VALUES ('072101005'),('072101006');

在列的列表中未给出名称的列都将赋予默认值。

同样也可以用 col_name = value 的形式给出列和值。

例 3-49　mysql>INSERT INTO Student SET Sno= '072101007';

在 SET 子句中未命名的行都赋予一个默认值。

注意，使用这种形式的 INSERT 语句不能插入多行。用 SELECT 语句查看当前表中内容：

```
mysql> SELECT * FROM Student;
```

```
+-----------+--------+------+------+--------+
| Sno       | Sname  | Ssex | Sage | Sdept  |
+-----------+--------+------+------+--------+
| 072101001 | 张三   | 男   |   19 | 信息系 |
| 072101002 | 李四   | 女   |   20 | 信息系 |
| 072101003 | 王五   | 男   |   20 | 信息系 |
| 072102001 | 李晨浩 | 男   |   21 | 物理系 |
| 072102002 | 刘敏菲 | 女   |   19 | 物理系 |
| 072102003 | 张勇云 | 男   |   20 | 物理系 |
| 072101004 | NULL   | NULL | NULL | NULL   |
| 072101005 | NULL   | NULL | NULL | NULL   |
| 072101006 | NULL   | NULL | NULL | NULL   |
| 072101007 | NULL   | NULL | NULL | NULL   |
+-----------+--------+------+------+--------+
```

【特别提示】一个 expression 可以引用在一个值表先前设置的任何列。例如，可以这样:

```
mysql> INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

但不能这样：

```
mysql> INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

2. 使用 INSERT…SELECT 语句插入从其他表选择的行

在前面学习创建表时，知道可以使用 SELECT 从其他表来直接创建表，甚至可以同时复制数据记录。如果已经拥有了一个表，同样可以从 SELECT 语句的配合中获益。

从其他表中录入数据，例如：

```
mysql>INSERT INTO tbl_name1(col1,col2) SELECT col3,col4 FROM tbl_name2;
```

也可以略去目的表的列表，如果每一列都有数据录入。

```
mysql>INSERT INTO tbl_name1 SELECT col3,col4 FROM tbl_name2;
```

INSERT INTO … SELECT 语句满足下列条件：

（1）查询不能包含一个 ORDER BY 子句。

（2）INSERT 语句的目的表不能出现在 SELECT 查询部分的 FROM 子句，因为 ANSI SQL 禁止从正在插入的表中 SELECT。

例 3-50　对每一个系，求学生的平均年龄，并把结果存入表 Deptage 中。

首先在数据库中建立一个新表，其中一列存放系名，另一列存放相应的学生平均年龄：

```
mysql>CREATE TABLE Deptage
     (
      Sdept char(15) UNIQUE,
      Avgage SMALLINT
     );
```

然后对 Student 表按系分组求平均年龄，再把系名和平均年龄存入新表中：

```
mysql>INSERT INTO Deptage(Sdept,Avgage)
      SELECT Sdept,AVG(Sage)
      FROM Student
      GROUP BY sdept;
```

用 SELECT 查看结果：

```
mysql> SELECT * FROM deptage;
```

```
+--------+--------+
| Sdept  | Avgage |
+--------+--------+
| NULL   | NULL   |
| 信息系 |     20 |
| 物理系 |     20 |
+--------+--------+
```

3．使用 REPLACE、REPLACE…SELECT 语句插入

REPLACE 功能与 INSERT 完全一样，不同之处在于 REPLACE 首先去查找需要插入的数据中是否在原表已经存在唯一索引的记录，如果已经存在则先把原表中的有关记录删除后再插入。对于这种情况，INSERT 语句的表现是产生一个错误。

REPLACE 语句也可以与 SELECT 相配合，所以前面的内容完全适合 REPALCE。

应该注意的是，由于 REPLACE 语句可能改变原有的记录，因此使用时要慎重。

例 3-51　对每一个系，求学生的平均年龄，并把结果存入表 Deptage 中。

```
mysql>REPLACE Deptage(Sdept,Avgage)
      SELECT Sdept,AVG(Sage)
      FROM Student
      GROUP BY sdept;
```

用 SELECT 查看结果：

```
mysql> SELECT * FROM deptage;
```

```
+---------+--------+
| Sdept   | Avgage |
+---------+--------+
| NULL    | NULL   |
| 信息系  |     20 |
| 物理系  |     20 |
| NULL    | NULL   |
+---------+--------+
4 rows in set
```

4．使用 LOAD 语句批量导入数据

本章的前面讨论如何使用 SQL 向一个表中插入数据。但是，如果需要向一个表中添加许多条记录，使用 SQL 语句插入数据是很不方便的。幸运的是，MySQL 提供了一些方法用于批量录入数据，使得向表中添加数据变得容易。

【特别提示】*LOAD 语句是 MySQL 所特有的，并不属于 ANSI SQL 语法，初学者可以先跳过本节。*

（1）基本语法

语法：LOAD DATA [LOCAL] INFILE 'file_name.txt' [REPLACE | IGNORE] INTO TABLE tbl_name

LOAD DATA INFILE 语句从一个文本文件中以很快的速度读入一个表中。如果指定 LOCAL 关键词，从客户主机读文件。如果 LOCAL 没指定，文件必须位于服务器上。

当读取位于服务器上的文本文件时，文件必须处于数据库目录或可被所有人读取。另外，为了对服务器上文件使用 LOAD DATA INFILE，在服务器主机上必须有相应的存取权限。

REPLACE 和 IGNORE 关键词控制对现有记录的主键重复处理。如果指定 REPLACE，新行将代替主键唯一的现有行；如果指定 IGNORE，跳过主键重复行的输入；如果不指定任何一个选项，当找到重复键时，出现一个错误，并且文本文件的余下部分被忽略。

如果使用 LOCAL 关键词从一个本地文件装载数据，服务器没有办法在操作的当中停止

文件的传输，因此默认值为 IGNORE。

（2）文件的搜索原则

当在服务器主机上寻找文件时，服务器使用下列规则：

- 如果给出一个绝对路径名，服务器使用该路径名。
- 如果给出一个或多个相对路径名，服务器相对服务器的数据目录搜索文件。
- 如果给出一个没有指定路径的文件名，服务器在当前数据库的数据库目录寻找文件。

注意这些规则意味着像“./myfile.txt”这样的文件是从服务器的数据目录读取，而作为“myfile.txt”这样的文件是从当前数据库的数据库目录下读取。也要注意，对于下列哪些语句，对 db1 文件从数据库目录读取，而不是 db2：

```
mysql> USE db1;
mysql> LOAD DATA INFILE "./data.txt" INTO TABLE db2.my_table;
```

（3）FIELDS 和 LINES 子句的语法

如果指定一个 FIELDS 子句，它的子句(TERMINATED BY、[OPTIONALLY] ENCLOSED BY 和 ESCAPED BY)也是可选的，不过，必须至少指定它们中的一个。

如果没有指定一个 FIELDS 子句，默认值相当于：

FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'

如果没有指定一个 LINES 子句，默认值相当于：

LINES TERMINATED BY '\n'

换句话说，默认值导致读取输入时，LOAD DATA INFILE 表现如下：

- 在换行符处寻找行边界。
- 在定位符处将行分成字段。
- 不要期望字段由任何引号字符封装。
- 将由“\”开头的定位符、换行符或“\”解释为字段值的部分字符。

LOAD DATA INFILE 能被用来读取从外部来源获得的文件。例如，以 dBASE 格式的文件将有由逗号分隔并用双引号包围的字段。如果文件中的行由换行符终止，下面显示的命令说明将用来装载文件的字段和行处理选项。

例 3-52　已知在一个 TXT 文本文件 data.txt 中（UTF-8 格式）有如下内容：

"0001","大学英语","1001","12",

"0002","高等数学","1002","6",

"0003","马克思主义哲学","1003","2",

"0004","邓小平理论概论","1004","2",

现要从 c:\data.txt 文件中导入表 course 中：

```
mysql> LOAD DATA INFILE 'c:\data.txt' INTO TABLE course
       FIELDS TERMINATED BY ',' ENCLOSED BY '"'
       LINES TERMINATED BY '\n';
mysql> SELECT * FROM course;
```

```
+------+--------------+------+---------+
| Cno  | Cname        | Cpno | Ccredit |
+------+--------------+------+---------+
| 0001 | 大学英语     | 1001 |      12 |
| 0002 | 高等数学     | 1002 |       6 |
| 0003 | 马克思主义哲学 | 1003 |       2 |
| 0004 | 邓小平理论概论 | 1004 |       2 |
+------+--------------+------+---------+
4 rows in set
```

任何字段或行处理选项可以指定一个空字符串（''）。如果不是空，FIELDS [OPTIONALLY] ENCLOSED BY 和 FIELDS ESCAPED BY 值必须是一个单个字符。FIELDS TERMINATED BY 和 LINES TERMINATED BY 值可以是超过一个字符。例如，写入由回车换行符对（CR+LF）终止的行，或读取包含这样行的一个文件，指定一个 LINES TERMINATED BY '\r\n'子句。

FIELDS [OPTIONALLY] ENCLOSED BY 控制字段的包围字符。对于输出(SELECT ... INTO OUTFILE)，如果省略 OPTIONALLY，所有的字段由 ENCLOSED BY 字符包围。

如果指定 OPTIONALLY，ENCLOSED BY 字符仅被用于类型为 CHAR 和 VARCHAR 的字段。一个字段值中的 ENCLOSED BY 字符的出现通过用 ESCAPED BY 字符作为其前缀来转义。要注意，如果指定一个空 ESCAPED BY 值，可能产生不能被 LOAD DATA INFILE 正确读出的输出。例如，如果转义字符为空，上面显示的输出显示如下。注意到在第 4 行的第二个字段包含跟随引号的一个逗号，它将终止字段导入：

"0001","大学英语","1001","12",
"0002","高等数学","1002","6",
"0003","马克思主义哲学","1003","2",
"0004","邓小平理论概论","1004","2",

【特别提示】用 Windows 自带的记事本可以建立一个文本文件，并可输入以上内容，但默认的记事本建立的文件是以 ANSI 码存放的，可在记事本的“文件”菜单中选择“另存为”命令，然后在“另存为”对话框中的编码一栏选择 UTF-8 选项即可存为 UTF-8 格式的文本文件，如图 3-6 所示。



图 3-6　data.txt 另存为 UTF-8 编码时的对话框

例 3-53　已知在一个 TXT 文本文件 data2.txt 中（UTF-8 格式）有如下内容：

"072101001","0001","92",
"072101001","0002","85",
"072101001","0003","88",
"072101002","0002","90",
"072101002","0003","80",
现要从 c:\data2.txt 文件中导入表 sc 中：

```
mysql> LOAD DATA INFILE 'c:\data2.txt' INTO TABLE sc
       FIELDS TERMINATED BY ',' ENCLOSED BY '"'
       LINES TERMINATED BY '\n';
mysql> SELECT * FROM sc;
```

```
+-----------+------+-------+
| Sno       | Cno  | Grade |
+-----------+------+-------+
| 072101001 | 0001 |    92 |
| 072101001 | 0002 |    85 |
| 072101001 | 0003 |    88 |
| 072101002 | 0002 |    90 |
| 072101002 | 0003 |    80 |
+-----------+------+-------+
5 rows in set
```

3.4.2　修改数据

修改操作语句的一般格式为：
UPDATE tbl_name SET 要更改的列
WHERE 要更新的记录
这里的 WHERE 子句是可选的，因此如果不指定，表中的每个记录都被更新。

1．修改某一个字段的值

例 3-54　表 Student 中，学号为“072101004”的姓名没有指定，可以这样修改这个记录：

```
mysql> UPDATE Student SET Sname="方月红" WHERE Sno="072101004" ;
```

2．修改多个字段的值

例 3-55　表 Student 中，修改学号为“072101004”的性别、年龄、系别：

```
mysql> UPDATE Student
       SET ssex="女",sage=20,Sdept="信息系"
       WHERE Sno="072101004" ;
```

修改的结果如下：

```
mysql> SELECT * FROM Student;
```

```
+-----------+--------+------+------+--------+
| Sno       | Sname  | Ssex | Sage | Sdept  |
+-----------+--------+------+------+--------+
| 072101001 | 张三   | 男   |   19 | 信息系 |
| 072101002 | 李四   | 女   |   20 | 信息系 |
| 072101003 | 王五   | 男   |   20 | 信息系 |
| 072102001 | 李晨浩 | 男   |   21 | 物理系 |
| 072102002 | 刘敏菲 | 女   |   19 | 物理系 |
| 072102003 | 张勇云 | 男   |   20 | 物理系 |
| 072101004 | 方月红 | 女   |   20 | 信息系 |
| 072101005 | NULL   | NULL | NULL | NULL   |
| 072101006 | NULL   | NULL | NULL | NULL   |
| 072101007 | NULL   | NULL | NULL | NULL   |
+-----------+--------+------+------+--------+
10 rows in set
```

3. 带子查询的修改语句

例 3-56 修改所有成绩表 sc 中没有任何成绩且性别为 NULL 记录中 sage、ssex、sdept 分别改为 20、“男”、“信息系”：

```
mysql> UPDATE Student
       SET ssex="男",sage=20,sdept="信息系"
       WHERE ISNULL(Sname) AND NOT EXISTS
          (SELECT  DISTINCT Sno
          FROM sc
          WHERE Student.Sno=sc.Sno);
```

修改的结果如下：

```
mysql> SELECT * FROM Student;
```

```
+-----------+--------+------+------+--------+
| Sno       | Sname  | Ssex | Sage | Sdept  |
+-----------+--------+------+------+--------+
| 072101001 | 张三   | 男   |   19 | 信息系 |
| 072101002 | 李四   | 女   |   20 | 信息系 |
| 072101003 | 王五   | 男   |   20 | 信息系 |
| 072102001 | 李晨浩 | 男   |   21 | 物理系 |
| 072102002 | 刘敏菲 | 女   |   19 | 物理系 |
| 072102003 | 张勇云 | 男   |   20 | 物理系 |
| 072101004 | 方月红 | 女   |   20 | 信息系 |
| 072101005 | NULL   | 男   |   20 | 信息系 |
| 072101006 | NULL   | 男   |   20 | 信息系 |
| 072101007 | NULL   | 男   |   20 | 信息系 |
+-----------+--------+------+------+--------+
10 rows in set
```

3.4.3　删除数据

DELETE 语句一般格式：

DELETE FROM tbl_name

WHERE 要删除的记录

WHERE 子句指定哪些记录应该删除。它是可选的，但如果不选，将会删除所有的记录。这意味着最简单的 DELETE 语句也是最危险的。

1. 删除一条记录

例 3-57 删除学号为“072101005”的学生记录。

```
mysql> DELETE FROM Student
       WHERE Sno="072101005" ;
```

2. 删除整个表中记录

例 3-58　清空整个表 Student：

```
mysql>DELETE FROM Student;
```

【特别提示】这条 DELETE 语句将使 Student 成为空表，需慎重使用。

3. 带子查询的删除语句

例 3-59　删除在表 sc 中不存在成绩的所有学生记录：

```
mysql>  DELETE
        FROM Student
        WHERE NOT EXIST
            (SELECT DISTINCT Sno
            FROM sc
            WHERE Student.Sno=sc.Sno);
```

修改的结果如下：

```
mysql> SELECT * FROM Student;
```

Sno	Sname	Ssex	Sage	Sdept
072101001	张三	男	19	信息系
072101002	李四	女	20	信息系

3.5　数据控制语言

由 DBMS 提供统一的数据控制功能是数据库系统的特点之一。SQL 中数据控制功能包括事务管理功能和数据保护功能，即数据库的恢复、并发控制；数据库的安全性和完整性控制。

SQL 语言定义完整性约束条件的功能主要体现在 CREATE TABLE 语句和 ALTER TABLE 中，可以在这些语句中定义关键字（码）、取唯一的列、不允许空值的列、外关键字（外码）及其他一些约束条件。

SQL 语言也提供了并发控制及恢复的功能，支持事务、提交、回滚等概念，这些将在后续章节中有进一步介绍。这里主要讨论 SQL 语言的安全性控制功能。数据库的安全性是指保护数据库，防止不合法使用所造成的数据泄露和破坏。数据库系统保证数据安全性的主要措施是进行存取控制，即规定不同用户对于不同数据对象所允许执行的操作，并控制各用户只能存取有权存取的数据。

某个用户对某类数据具有何种操作权力是个政策问题而不是技术问题，数据库管理系统的功能是保证这些决策的执行。因此，DBMS 必须具有以下功能：

（1）用户或 DBA 把授权决定告知系统，这是由 SQL 的 GRANT 和 REVOKE 语句来完成的。

（2）DBMS 把授权的结果存入数据字典。

（3）当用户提出操作请求时，DBMS 根据授权情况进行检查，以决定是否执行操作请求。

不同的用户对不同的数据应具有何种操作权力，是由数据库管理员（DBA）和表的建立者（即表的属主）根据具体情况决定的，SQL 语言则为 DBA 和表的属主定义与回收这种权力提供了手段。

MySQL 的安全系统是很灵活的，它允许以多种不同方式设置用户权限。一般可使用标准的 SQL 语句 GRANT 和 REVOKE 语句修改控制客户访问的授权表。客户对 MySQL 数据库的访问由授权表内容来控制。这些表位于 MySQL 数据库中，并在第一次安装 MySQL 的过程中初始化。授权表共有 5 个表：user、db、host、tables_priv 和 columns_priv。

【特别提示】并发控制指的是当多个用户并发地对数据库进行操作时，对它们加以控制、协调，以保证并发操作正确执行，并保持数据库的一致性。恢复指的是当发生各种类型的故障，使数据库处于不一致状态时，将数据库恢复到一致状态的功能。

3.5.1　授权

GRANT 语句的一般格式：

```
GRANT priv_type (columns)
ON what
TO user IDENTIFIED BY "password"
WITH GRANT OPTION
```

功能：将对指定操作对象的指定操作权限授予指定的用户。

1．priv_type 权限类型

对不同类型的操作对象有不同的操作权限，常见的操作权限如表 3-13 所示。

表 3-13　不同对象类型允许的操作权限

对　象	对 象 类 型	操作权限（priv_type）
属性列	TABLE	SELECT,INSERT,UPDATE,DELETE,ALL RIVIEGES
视图	TABLE	SELECT,INSERT,UPDATE,DELETE,ALL RIVIEGES
基本表	TABLE	SELECT,INSERT,UPDATE,DELETE,ALTER,INDEX,ALL PRIVIEGES
数据库	DATABASE	CREATE TAB

GRANT 语句中 columns 权限适用的列是可选的，如果多于一个列，则用逗号分开。

2．ON what 权限级别

what 对应有 4 个权限应用的级别。GRANT 允许系统授权 MySQL 用户的权利如下：

（1）全局级别

全局权限作用于一个给定服务器上的所有数据库。这些权限存储在 mysql.user 表中。可

以通过使用 ON *.*语法设置全局权限。

（2）数据库级别

数据库权限作用于一个给定数据库的所有表。这些权限存储在 mysql.db 和 mysql.host 表中。通过使用 ON db_name.*语法设置数据库权限。如果指定 ON *并且有一个当前数据库，将为该数据库设置权限。

【特别提示】如果指定 ON *而没有一个当前数据库，将影响全局权限。

（3）表级别

表权限作用于一个给定表的所有列。这些权限存储在 mysql.tables_priv 表中。可以通过 ON tbl_name，为具体的表名设置权限。

（4）列级别

列权限作用于在一个给定表的单个列。这些权限存储在 mysql.columns_priv 表中。可以通过指定一个 columns 子句将权限授予特定的列，同时要在 ON 子句中指定具体的表。

3. user 使用权限的用户

MySQL 支持以 username@host 格式指定 user 值。如果想要指定一个特殊字符的一个 user 字符串（例如“-”），或一个包含特殊字符或通配符的 host 字符串（例如“%”），可以用括号括起用户或主机名称（例如，'test-user'@'test-hostname'）。可以在主机名中指定通配符。例如，user1@"%.loc.gov"适用于在 loc.gov 域中任何主机的 user1，并且 user2 @ "144.155.166.% "适用于在 IP 段为 144.155.166 类 C 子网中任何主机的 user2。

user 相当于 username@"%"。如果允许匿名用户连接 MySQL 服务器，应该增加所有本地用户如 username@localhost；否则，当用户试图从本地机器上登录到 MySQL 服务器时，对于 mysql.user 表中的本地主机的匿名用户条目将被使用。匿名用户通过插入有 User="的条目到 mysql.user 表中来定义。

4. password 子句

password 分配给该用户的口令也是可选的。

如果创建一个新用户或全局授予权限，用户的口令将被设置为由 IDENTIFIED BY 子句指定的口令，如果用户已经有了一个口令，它将被新的代替。

要注意的是，如果建立一个新用户但不指定一个 IDENTIFIED BY 子句，用户没有口令，这是不安全的。

5. WITH GRANT OPTION 子句

WITH GRANT OPTION 子句是可选的，它给予用户有授予其他用户在指定的权限水平上的任何权限的能力。应该谨慎对待授予 grant 权限的用户，因为具有不同权限的两个用户也许能合并权限。

例 3-60　把查询 Student 表的权限授给用户 U1：

```
mysql>GRANT SELECT
      ON TABLE Student
      TO U1;
```

例 3-61　把对 mis 数据库所有表的全部操作权限授给用户 U2 和 U3：

```
mysql>GRANT ALL PRIVILEGES
      ON mis.*
      TO U2,U3;
mysql>SELECT host,user,password FROM mysql.user;
```

例 3-62　把对表 SC 的 INSERT 权限授予 U4 用户，并允许再将此权限授予其他用户：

```
mysql>GRANT INSERT
      ON TABLE SC TO U4
      WITH GRANT OPTION;
```

执行后，U4 不仅拥有了对表 SC 的 INSERT 权限，还可以传播此权限：

```
mysql>GRANT INSERT
      ON TABLE SC TO U5
      WITH GRANT OPTION;
```

同样，U5 还可以将此权限授予 U6：

```
mysql>GRANT INSERT
ON TABLE SC TO U6;
```

但 U6 不能再传播此权限，即 U4-> U5-> U6。

例 3-63　在本机 localhost 上建立一个超级用户 admin，密码为“111”。

```
mysql>  GRANT ALL
        ON *.* TO admin@localhost IDENTIFIED BY "111"
        WITH GRANT OPTION;
```

查看 mysql.user 表中的 host，user，password 列：

```
mysql>  SELECT host,user,password
        FROM mysql.user;
```

```
+-----------+-------+-------------------------------------------+
| host      | user  | password                                  |
+-----------+-------+-------------------------------------------+
| localhost | root  | *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B |
| %         | U2    |                                           |
| %         | U3    |                                           |
| localhost | admin | *832EB84CB764129D05D498ED9CA7E5CE9B8F83EB |
| %         | U1    |                                           |
| %         | U4    |                                           |
| %         | U5    |                                           |
| %         | U6    |                                           |
+-----------+-------+-------------------------------------------+
8 rows in set
```

可以测试刚才建立的新账号 admin，使用 Navicat 新建一个连线，在“连线”对话框中使用者名称输入“admin”，密码输入“111”，连线名称输入“localhost2”，并单击“连线测试”按钮，如果显示“连接成功”则说明已经正常，如图 3-7 所示。

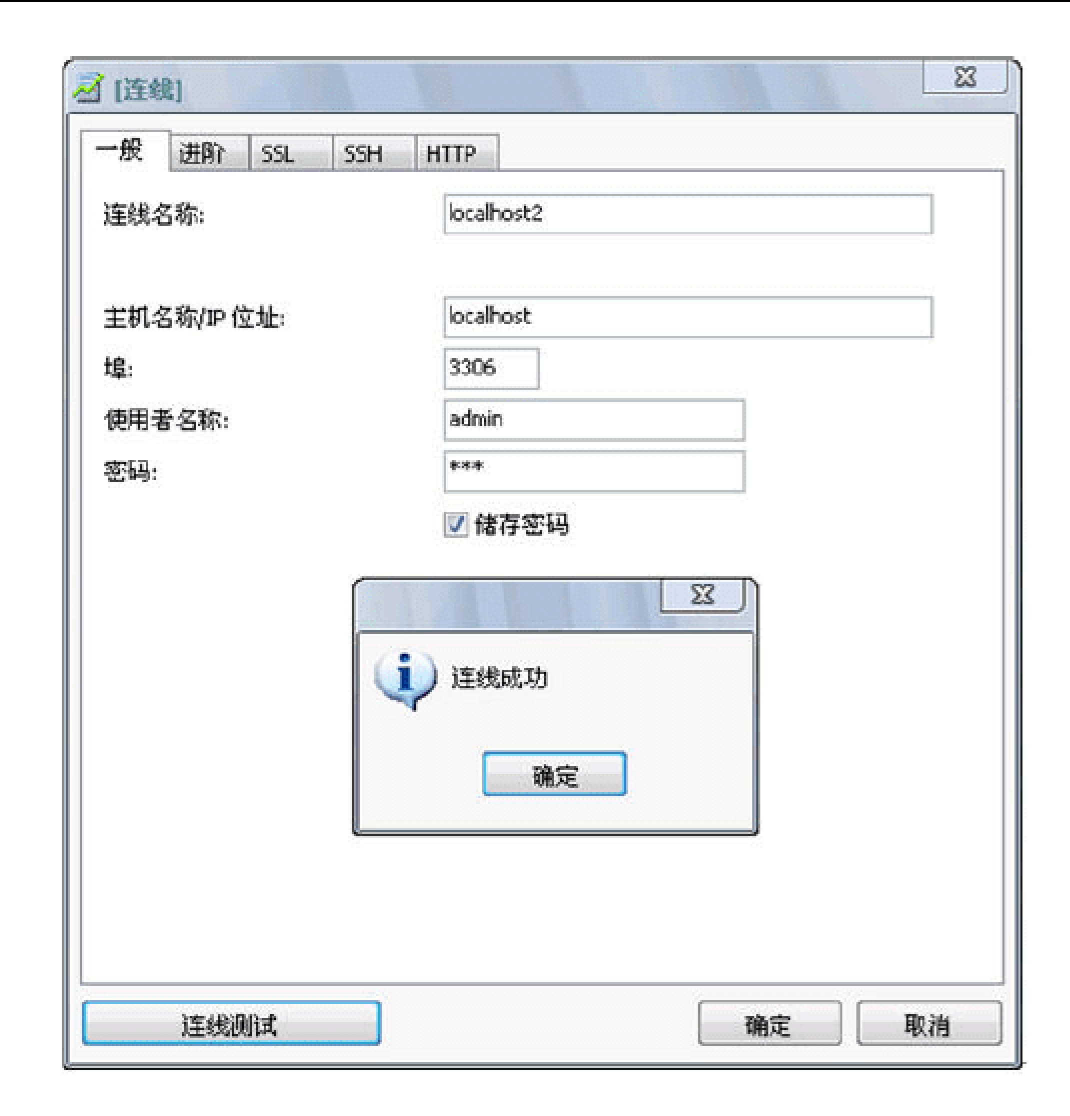


图 3-7　新建一个连线验证 admin 用户

3.5.2　回收权限

回收某个用户的权限，可使用 REVOKE 语句。除了要用 FROM 替换 TO 并且没有 IDENTIFIED BY 或 WITH GRANT OPTION 子句外，其他与 GRANT 语法类似。

REVOKE 语句的一般格式：

```
REVOKE privileges (columns)
ON what
FROM user;
```

user 部分必须与想要取消其权限的用户的原始 GRANT 语句的 user 部分相匹配。

Privileges 部分不需要匹配，REVOKE 可以只回收部分权限。

【特别提示】 REVOKE 语句只删除权限，不删除用户。即使回收了用户的所有权限，用户的项仍然保留在 user 表中。这意味着该用户仍然可以连接到服务器上。要删除整个用户，必须用 DELETE 语句将该用户的记录从 user 表中直接删除。

例 3-64　用 REVOKE 回收 admin 的所有授权：

```
mysql>REVOKE ALL ON *.* FROM admin@localhost;
```

但是，admin@localhost 用户的条目仍旧留在 user 表中，可以查看：

```
mysql> SELECT host,user,password FROM mysql.user where user='admin';
+-----------+-------+-------------------------------------------+
| host      | user  | password                                  |
+-----------+-------+-------------------------------------------+
| localhost | admin | *832EB84CB764129D05D498ED9CA7E5CE9B8F83EB |
+-----------+-------+-------------------------------------------+
1 row in set
```

可见，SQL 提供了非常灵活的授权机制。DBA 拥有对数据库中所有对象的所有权限，并可以根据应用的需要将不同的权限授予不同的用户。用户对自己建立的基本表和视图拥

有全部的操作权限，并且可以用 GRANT 语句把其中某些权限授予其他用户。被授权的用户如果有“继续授权”的许可，还可以把获得的权限再授予其他用户。所有授予出去的权力在必要时又都可以用 REVOKE 语句收回。

3.6　嵌入式 SQL

SQL 语言提供了 3 种不同的使用方式：一种是在终端交互式方式下使用，前面介绍的就是作为独立语言由用户在交互环境下使用的 SQL 语言。第二种是将 SQL 语言嵌入到某种高级语言如 PL/SQL、COBOL、FORTRAN、C 中使用，利用高级语言的过程性结构来弥补 SQL 语言在实现复杂应用方面的不足，这种方式下使用的 SQL 语言称为嵌入式 SQL（Embedded SQL），而嵌入 SQL 的高级语言称为主语言或宿主语言。第三种是应用程序编程接口（API），如 ODBC/JDBC。当然这 3 种方式细节上会有许多差别，在程序设计的环境下，SQL 语句要做某些必要的扩充。

在嵌入式 SQL 中，为了能够区分 SQL 语句与主语言语句，所有 SQL 语句都必须加前缀 EXEC SQL。SQL 语句的结束标志则随主语言的不同而不同，例如在 PL/1 和 C 中以分号（;）结束，在 COBOL 中以 END-EXEC 结束。这样，以 C 或 PL/1 作为主语言的嵌入式 SQL 语句的一般形式为：

```
EXEC SQL<SQL 语句>;
```

3.6.1　嵌入式 SQL 语句与主语言之间的通信

将 SQL 嵌入到高级语言中混合编程，SQL 语句负责操纵数据库，高级语言语句负责控制程序流程。数据库工作单元与源程序工作单元之间通信主要包括：

（1）向主语言传递 SQL 语句的执行状态信息，使主语言能够据此控制程序流程，主要用 SQL 通信区（SQL Communication Area，SQLCA）实现。

（2）主语言向 SQL 语句提供参数，主要用主变量（host variable）实现。

（3）将 SQL 语句查询数据库的结果交主语言进一步处理，主要用主变量和游标（cursor）实现。

【特别提示】 嵌入式 SQL 语句中可以使用主语言的程序变量输入或输出数据，把 SQL 语句中使用的主语言程序变量简称为主变量。游标是系统为用户开设的一个数据缓冲区，存放 SQL 语句的执行结果，每个游标都有一个名字。用户可以逐一获取记录，并赋给主变量，交由主语言进一步处理。

例 3-65　下面给出带有嵌入式 SQL 的一小段 C 程序。

```
EXEC SQL INCLUDE SQLCA; /*定义 SQL 通信区*/
EXEC SQL BEGIN DECLARE SECTION; /*说明主变量*/
     CHAR sno_t(9);
     CHAR sname_t(8);
```

```
  INT sage_t;
EXEC SQL END DECLARE SECTION;/*主变量说明结束*/
main()
{
  EXEC SQL DECLARE C1 CURSOR FOR /*游标操作(定义)*/
      SELECT Sno,Sname,Sage
      FROM Student;
  EXEC SQL OPEN C1; /*游标操作(打开)*/
  for(;;)
  {
      EXEC SQL FETCH C1 INTO :sno_t, :sname_t, :sage_t;
         /* 游标操作（推进游标指针,将当前数据放入主变量 ）*/
      if (sqlca.sqlcode <> SUCCESS)
         break; /* 利用 SQLCA 中的状态信息,决定何时退出循环*/
    printf("姓名%s, 年龄: %d", : sno_t, : sage_t);
  }
  EXEC SQL CLOSE C1;/*游标操作（关闭游标）*/
}
```

3.6.2　不用游标的 SQL 语句

（1）说明性语句

说明性语句是专为嵌入式 SQL 中说明主体变量而设置的，主要有两条语句：

```
EXEC SQL BEGIN DECLARE SECTION;
     /*主变量说明*/
EXEC SQL END DECLARE SECTION;
```

两条语句必须配对出现，相当于一个括号，两条语句中间是主体变量的说明（见例 3-65）。

（2）数据定义语句

例 3-66　建立一个学生表 Student。

```
EXEC SQL CREATE TABLE Student
        (Sno CHAR(5) NOT NULL UNIQUE,
         Sname CHAR(20),
         Ssex CHAR(1),
         Sage INT,
         Sdept CHAR(15));
```

例 3-67　删除学生表 Student。

```
EXEC SQL DROP TABLE Student;
```

（3）数据控制语句

例 3-68　把查询 Student 表权限授给用户 U1。

```
EXEC SQL GRANT SELECT ON TABLE Student  TO U1;
```

（4）查询结果为单记录的 SELECT 语句

在嵌入式 SQL 中，查询结果为单记录的 SELECT 语句需要用 INTO 子句指定查询结果存放地点。该语句的一般格式为：

```
EXEC SQL SELECT [ALL|DISTINCT] <目标列表达式>
        [,<目标列表达式>]...
        INTO <主变量>[<指示变量>]
        [,<主变量>[<指示变量>]]...
        FROM <表名或视图名>[,<表名或视图名>] ...
        [WHERE <条件表达式>]
        [GROUP BY <列名 1>[HAVING <条件表达式>]]
        [ORDER BY <列名 2> [ASC|DESC]];
```

例 3-69　根据学生学号查询学生信息（假设学号已存入主变量 sno_t）。

```
EXEC SQL SELECT Sno,Sname,Ssex,Sage,Sdept
        INTO :Hsno,:Hname,:Hsex,:Hage,:Hdept
        FROM Student
        WHERE Sno=:sno_t;
```

例 3-70　查询某个学生选修某门课程的成绩（假设学号已存入主变量 sno_t，课程号赋给了主变量 cno_t）。

```
EXEC SQL SELECT Sno, Cno, Grade
        INTO :Hsno, :Hcno, :Hgrade:Gradeid
        FROM SC
        WHERE Sno=:sno_t AND Cno=:cno_t;
```

（5）非 CURRENT 形式的 UPDATE 语句

在 UPDATE 语句中，SET 子句和 WHERE 子句中均可以使用主变量，其中 SET 子句中还可以使用指示变量。

例 3-71　将全体学生课程号为 1 的考试成绩减少 score_t（主变量，存储减分额度）。

```
EXEC SQL UPDATE SC
        SET Grade=Grade-:score_t
        WHERE Cno='1';
```

（6）非 CURRENT 形式的 DELETE 语句

DELETE 语句的 WHERE 子句中可以使用主变量指定删除条件。

例 3-72　删除某个学生所有选修课程记录（学生姓名已存入主变量 sname_t）。

```
EXEC SQL DELETE
        FROM SC
        WHERE Sno=
           (SELECT Sno
            FROM Student
            WHERE Sname=:sname_t);
```

（7）INSERT 语句

INSERT 语句的 VALUES 子句中可以使用主变量和指示变量。

例 3-73　某个学生新选修了某门课程，将有关记录插入 sc 表中（假设学号已存入主变量 sno_t，课程号已赋给主变量 cno_t）。

```
gradeid=-1;
EXEC SQL INSERT
  INTO SC(Sno, Cno, Grade)
  VALUES(:sno_t, :cno_t, :gr:gradeid);
```

【特别提示】将指示变量 gradeid 赋一个负值后，无论主变量 gr 为何值，都会将 Grade 字段值置为空。

3.6.3　使用游标的 SQL 语句

（1）查询结果为多条记录的 SELECT 语句

一般情况下，SELECT 语句查询结果都是多条记录，而高级语言一次只能处理一条记录，因此需要用游标机制，将多条记录一次一条送至宿主程序处理，从而把对集合的操作转换为对单个记录的处理。

例 3-74　查询某个系全体学生的信息（要查询的系名由用户在程序运行过程中指定，放在主变量 deptname_t 中）。

```
...
EXEC SQL BEGIN DECLARE SECTION;
...
    /* 说明主变量 deptname_t,HSno,HSname,HSsex,HSage 等*/
...
EXEC SQL END DECLARE SECTION;
...
gets(deptname_t);    /* 为主变量 deptname_t 赋值 */
...
EXEC SQL DECLARE SX CURSOR FOR
    SELECT Sno, Sname, Ssex, Sage
    FROM Student
     WHERE SDept=:deptname_t;  /* 说明游标 */
EXEC SQL OPEN SX          /* 打开游标 */
WHILE(1)  /* 用循环结构逐条处理结果集中的记录 */
{
    EXEC SQL FETCH SX INTO:HSno,:HSname,:HSsex,:Hsage;
    /* 游标指针向前推进一行，然后从结果集中取当前行，送相应主变量 */
    if (sqlca.sqlcode <> SUCCESS)
        break;
...
}
EXEC SQL CLOSE SX;      /* 关闭游标 */
...
```

（2）CURRENT 形式的 UPDATE 语句和 CURRENT 形式的 DELETE 语句

UPDATE 语句和 DELETE 语句都是集合操作，如果只想修改或删除其中某条记录，则需要用带游标的 SELECT 语句查出所有满足条件的记录，从中进一步找出要修改或删除的记录，然后用 CURRENT 形式的 UPDATE 和 DELETE 语句修改或删除。

例 3-75　查询某个系全体学生的信息（要查询的系名由主变量 deptname_t 指定），然后根据用户的要求修改其中某些记录的年龄字段。

```
...
EXEC SQL BEGIN DECLARE SECTION;
...
   /* 说明主变量 deptname_t、Hsno、Hsname、Hssex、HSage 等*/
...
EXEC SQL END DECLARE SECTION;
...
   gets(deptname_t);   /* 为主变量 deptname_t 赋值 */
...
EXEC SQL DECLARE SX CURSOR FOR
SELECT Sno, Sname, Ssex, Sage
FROM Student
   WHERE SDept=:deptname_t; /* 说明游标 */
   FOR UPDATE OF Sage;
EXEC SQL OPEN SX          /* 打开游标 */
WHILE(1)  /* 用循环结构逐条处理结果集中的记录 */
{
   EXEC SQL FETCH SX INTO:HSno,:HSname,:HSsex,:Hsage;
   /* 游标指针向前推进一行，然后从结果集中取当前行，送相应主变量 */
   if (sqlca.sqlcode <> SUCCESS)
     break;
...
printf("%s, %s, %s, %d",Sno,Sname,Ssex, Sage); /* 显示该记录 */
printf("UPDATE AGE?");      /* 问用户是否要修改 */
scanf("%c",&yn);
if (yn='y' or yn='Y')      /* 需要修改 */
{
   printf("INPUT NEW AGE: ");
   scanf("%d",&NEWAge);      /* 输入新的年龄值 */
   EXEC SQL UPDATE Student
     SET Sage=:NEWAge
     WHERE CURRENT OF SX;
     /* 修改当前记录的年龄字段 */
   }
   ...
}
EXEC SQL CLOSE SX;         /* 关闭游标 */
...
```

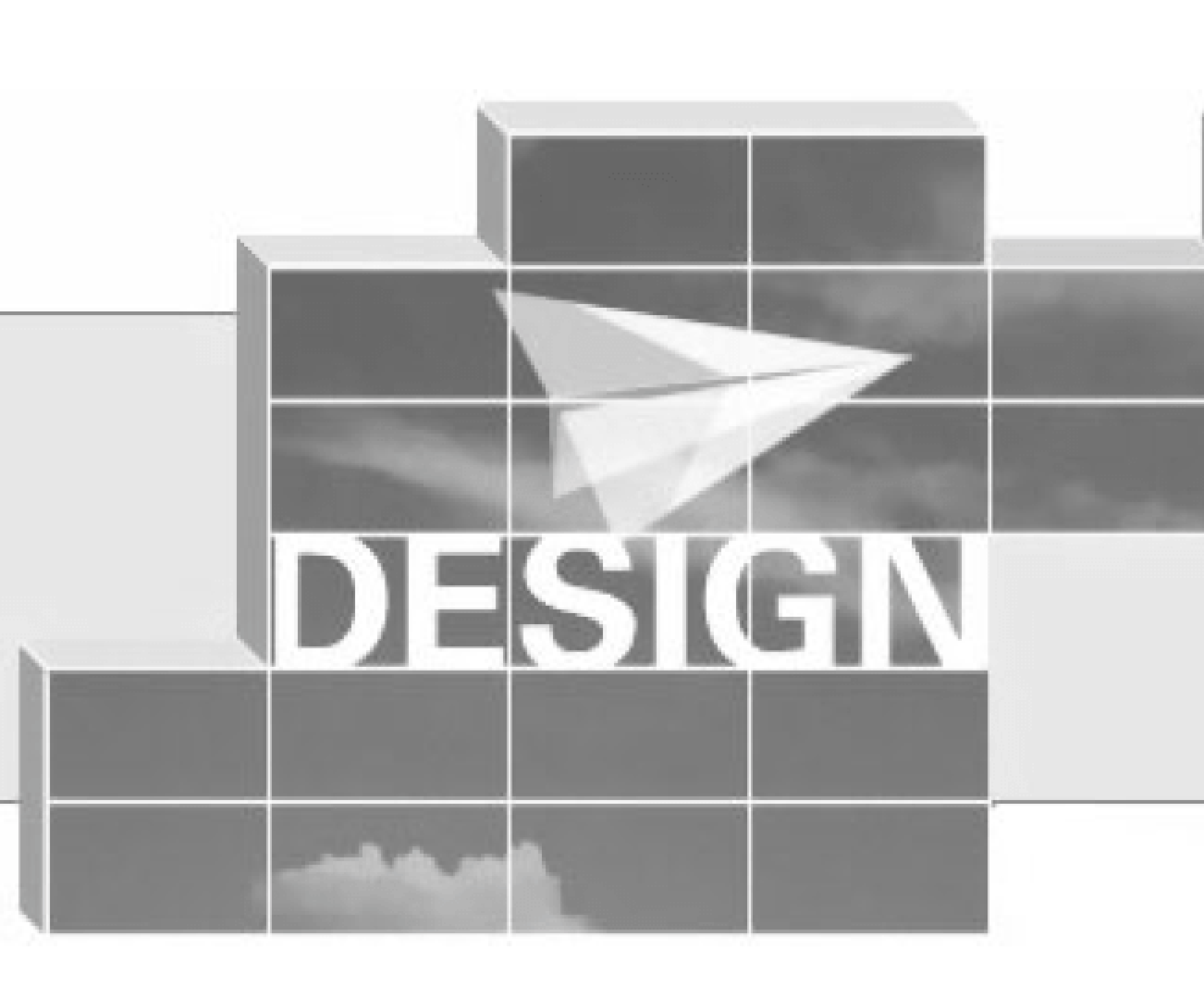
3.7　小　　结

本章系统而详尽地讲解了 SQL 语言。SQL 是关系数据库语言的工业标准。各个数据库厂商支持的 SQL 语言在遵循标准的基础上常常作不同的扩充或修改。本章介绍的是标准 SQL，因此，本章的绝对大部分例子应能在不同的系统如 MySQL、MSSQL、Oracle 等众多系统上运行，某些例子可能需要稍作修改才能运行。本章 3.1~3.5 节中所有例题均在 MySQL 5.0+ Navicat 8 for MySQL 中实现。

在讲解了 SQL 语言的同时进一步讲解了关系数据库系统的基本概念，使这些概念更加具体、丰富。

SQL 语言可以分为数据定义、数据查询、数据更新、数据控制 4 大部分。SQL 语言的数据查询功能是最丰富，也是最复杂的，读者应加强练习。

第 2 篇



苦练基本功

熟悉了 MySQL 开发工具和 SQL 语言后，接下来就需要学会如何熟练运用 MySQL 和 Java 来进行开发了。

本篇共有 5 章，主要内容包括数据库分析与设计、存储过程与触发器、JDBC 编程、Connector/J 的使用等内容。在本篇中将是一个苦练基本功的过程，在掌握了这些知识后，就能够为项目实战打下坚实的基础。

第 4 章　数据库分析与设计

本章主要介绍数据分析与设计知识，包括数据设计概述，数据库的设计范式，需求分析阶段、选择键和索引、数据完整性设计、表和字段的设计等数据库设计技巧，Power Designer 10 工作环境，最后介绍 Power Designer 中的正向工程与逆向工程。

4.1　数据设计概述

数据库设计（Database Design）是指对于一个给定的应用环境，构造最优的数据库模式，建立数据库及其应用系统，使之能够有效地存储数据，满足各种用户的应用需求（信息要求和处理要求）。

在数据库领域内，常常把使用数据库的各类系统统称为数据库应用系统。

4.1.1　数据库和信息系统

（1）数据库是信息系统的核心和基础，把信息系统中大量的数据按一定的模型组织起来，提供存储、维护、检索数据的功能，使信息系统可以方便、及时、准确地从数据库中获得所需的信息。

（2）数据库是信息系统的各个部分能否紧密地结合在一起以及如何结合的关键所在。

（3）数据库设计是信息系统开发和建设的重要组成部分。

（4）数据库设计人员应该具备的技术和知识介绍如下。

- 数据库的基本知识和数据库设计技术。
- 计算机科学的基础知识和程序设计的方法和技巧。
- 软件工程的原理和方法。
- 应用领域的知识。

4.1.2　数据库设计的特点

传统的软件工程忽视对应用中数据语义的分析和抽象，只要有可能就尽量推迟数据结构设计的决策，早期的数据库设计致力于数据模型和建模方法研究，忽视了对行为的设计，如图 4-1 所示。

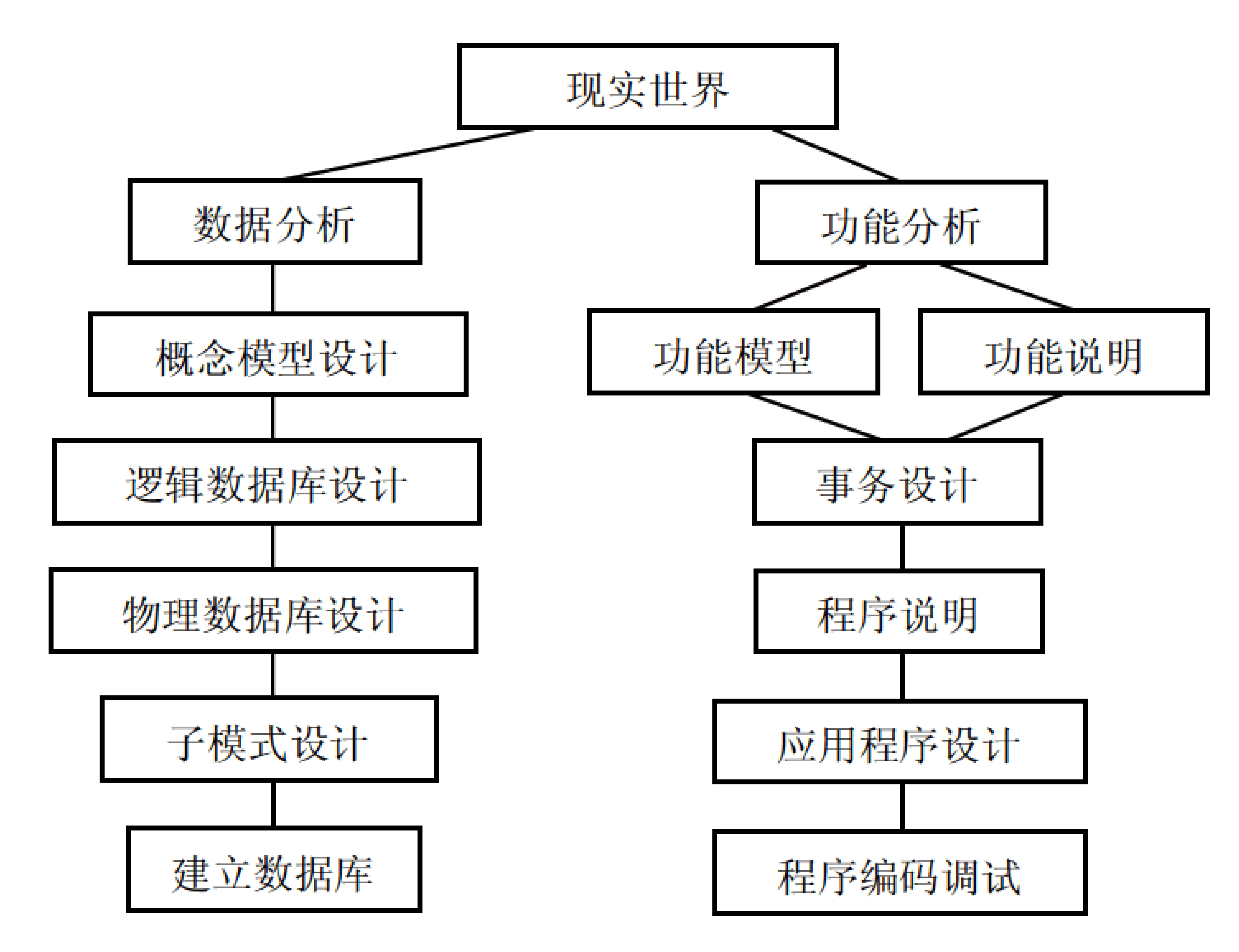


图 4-1　传统的软件工程的数据库设计

4.1.3　数据库设计的基本步骤

一般把数据库设计的过程分为 6 个阶段。

（1）需求分析阶段

准确了解与分析用户需求（包括数据与处理）是整个设计过程的基础，是最困难、最耗费时间的一步。

（2）概念结构设计阶段

该阶段是整个数据库设计的关键，通过对用户需求进行综合、归纳与抽象，形成一个独立于具体 DBMS 的概念模型。

（3）逻辑结构设计阶段

将概念结构转换为某个 DBMS 所支持的数据模型对其进行优化。

（4）数据库物理设计阶段

为逻辑数据模型选取一个最适合应用环境的物理结构（包括存储结构和存取方法）。

（5）数据库实施阶段

运用 DBMS 提供的数据语言、工具及宿主语言，根据逻辑设计和物理设计的结果建立数据库，编制与调试应用程序，组织数据入库，并进行试运行。

（6）数据库运行和维护阶段

数据库应用系统经过试运行后即可投入正式运行，在数据库系统运行过程中必须不断地对其进行评价、调整与修改。

在设计过程中把数据库的设计和对数据库中数据处理的设计紧密结合起来，将这两个方面的需求分析、抽象、设计，实现在各个阶段同时进行，相互参照、相互补充，以完善两方面的设计。

4.1.4　数据库各级模式的形成过程

（1）需求分析阶段：综合各个用户的应用需求。

（2）概念设计阶段：形成独立于机器特点，独立于各个 DBMS 产品的概念模式（E-R 图）。

（3）逻辑设计阶段：首先将 E-R 图转换成具体的数据库产品支持的数据模型，如关系模型，形成数据库逻辑模式；然后根据用户处理的要求、安全性的考虑，在基本表的基础上再建立必要的视图（View），形成数据的外模式。

（4）物理设计阶段：根据 DBMS 特点和处理的需要，进行物理存储安排，建立索引，形成数据库内模式。

4.2　数据库的设计范式

数据库的设计范式是数据库设计所需要满足的规范，满足这些规范的数据库是简洁的、结构明晰的，同时，不会发生插入（insert）、删除（delete）和更新（update）操作异常。反之则是乱七八糟，不仅给数据库的编程人员制造麻烦，而且面目可憎，可能存储了大量不需要的冗余信息。

第一范式（1NF）：数据库表中的字段都是单一属性的，不可再分。这个单一属性由基本类型构成，包括整型、实数型、字符型、逻辑型、日期型等。

例如，如下的数据库表是符合第一范式的：

字段 1	字段 2	字段 3	字段 4

而这样的数据库表是不符合第一范式的：

字段 1	字段 2	字段 3		字段 4
		字段 3.1	字段 3.2	

很显然，在当前的任何关系数据库管理系统（DBMS）中，“傻瓜”也不可能做出不符合第一范式的数据库，因为这些 DBMS 不允许把数据库表的一列再分成两列或多列。因此，想在现有的 DBMS 中设计出不符合第一范式的数据库都是不可能的。

第二范式（2NF）：数据库表中不存在非关键字段对任一候选关键字段的部分函数依赖（部分函数依赖指的是存在组合关键字中的某些字段决定非关键字段的情况），也即所有非关键字段都完全依赖于任意一组候选关键字。假定选课关系表为 SelectCourse(学号, 姓名, 年龄, 课程名称, 成绩, 学分)，关键字为组合关键字(学号, 课程名称)，因为存在如下决定关系：

(学号, 课程名称) → (姓名, 年龄, 成绩, 学分)

这个数据库表不满足第二范式，因为存在如下决定关系：

(课程名称) → (学分)

(学号) → (姓名, 年龄)

即存在组合关键字中的字段决定非关键字的情况。

由于不符合 2NF，这个选课关系表会存在如下问题。

（1）数据冗余

同一门课程由 n 个学生选修，"学分"就重复 n−1 次；同一个学生选修了 m 门课程，姓名和年龄就重复了 m−1 次。

（2）更新异常

若调整了某门课程的学分，数据表中所有行的"学分"值都要更新，否则会出现同一门课程学分不同的情况。

（3）插入异常

假设要开设一门新的课程，暂时还没有人选修。这样，由于还没有"学号"关键字，课程名称和学分也无法记录入数据库。

（4）删除异常

假设一批学生已经完成课程的选修，这些选修记录就应该从数据库表中删除。但与此同时，课程名称和学分信息也被删除了。很显然，这也会导致插入异常。

把选课关系表 SelectCourse 改为如下 3 个表：

- 学生：Student(学号, 姓名, 年龄)。
- 课程：Course(课程名称, 学分)。
- 选课关系：SelectCourse(学号, 课程名称, 成绩)。

这样的数据库表是符合第二范式的，消除了数据冗余、更新异常、插入异常和删除异常。

另外，所有单关键字的数据库表都符合第二范式，因为不可能存在组合关键字。

第三范式（3NF）：在第二范式的基础上，数据表中如果不存在非关键字段对任一候选关键字段的传递函数依赖则符合第三范式。所谓传递函数依赖，指的是如果存在"A→B→C"的决定关系，则 C 传递函数依赖于 A。因此，满足第三范式的数据库表应该不存在如下依赖关系：

关键字段→非关键字段 x→非关键字段 y

假定学生关系表为 Student(学号, 姓名, 年龄, 所在学院, 学院地点, 学院电话)，关键字为单一关键字"学号"，因为存在如下决定关系：

(学号)→ (姓名, 年龄, 所在学院, 学院地点, 学院电话)

这个数据库是符合 2NF 的，但是不符合 3NF，因为存在如下决定关系：

(学号)→(所在学院)→(学院地点, 学院电话)

即存在非关键字段"学院地点"、"学院电话"对关键字段"学号"的传递函数依赖。

它也会存在数据冗余、更新异常、插入异常和删除异常的情况，读者可自行分析得知。

把学生关系表分为如下两个表。

- 学生：(学号, 姓名, 年龄, 所在学院)。
- 学院：(学院, 地点, 电话)。

这样的数据库表是符合第三范式的，消除了数据冗余、更新异常、插入异常和删除异常。

鲍依斯-科得范式（BCNF）：在第三范式的基础上，数据库表中如果不存在任何字段对任一候选关键字段的传递函数依赖则符合第三范式。

假设仓库管理关系表为 StorehouseManage(仓库 ID, 存储物品 ID, 管理员 ID, 数量)，且有一个管理员只在一个仓库工作；一个仓库可以存储多种物品。这个数据库表中存在如下决定关系：

(仓库 ID, 存储物品 ID)→(管理员 ID, 数量)

(管理员 ID, 存储物品 ID)→(仓库 ID, 数量)

所以，(仓库 ID, 存储物品 ID)和(管理员 ID, 存储物品 ID)都是 StorehouseManage 的候选关键字，表中的唯一非关键字段为数量，它是符合第三范式的。但是，由于存在如下决定关系：

(仓库 ID)→(管理员 ID)

(管理员 ID)→(仓库 ID)

即存在关键字段决定关键字段的情况，所以其不符合 BCNF 范式。它会出现如下异常情况。

（1）删除异常

当仓库被清空后，所有"存储物品 ID"和"数量"信息被删除的同时，"仓库 ID"和"管理员 ID"信息也被删除了。

（2）插入异常

当仓库没有存储任何物品时，无法给仓库分配管理员。

（3）更新异常

如果仓库换了管理员，则表中所有行的管理员 ID 都要修改。

把仓库管理关系表分解为两个关系表。

- 仓库管理：StorehouseManage(仓库 ID, 管理员 ID)。
- 仓库：Storehouse(仓库 ID, 存储物品 ID, 数量)。

这样的数据库表是符合 BCNF 范式的，消除了删除异常、插入异常和更新异常。

4.3　数据库设计技巧

在信息系统的开发中，无论使用的是 Oracle、SQL Server、DB2 等大型数据库或者 MySQL、Access 等中小型数据库，数据库设计的重要性不言而喻。为设计出规范高效的数据库，有必要归纳总结数据库设计过程中的常用技巧。

4.3.1　需求分析阶段

（1）理解客户需求，询问用户如何看待未来需求变化。让客户解释其需求，而且随着开发的继续，还要经常询问客户保证其需求仍然在开发的目的之中。

（2）了解企业业务可以在以后的开发阶段节约大量的时间。

（3）重视输入输出。

在定义数据库表和字段需求（输入）时，首先应检查现有的或者已经设计出的报表、查询和视图（输出），以决定为了支持这些输出哪些是必要的表和字段。

例如，假如客户需要一个报表按照邮政编码排序、分段和求和，要保证其中包括了单独的邮政编码字段，而不要把邮政编码糅进地址字段中。

（4）创建数据字典和 ER 图表。

ER 图表和数据字典可以让任何了解数据库的人都明确如何从数据库中获得数据。ER 图对表明表之间的关系很有用，而数据字典则说明了每个字段的用途以及任何可能存在的别名。对 SQL 表达式的文档化来说这是完全必要的。

（5）定义标准的对象命名规范。

数据库各种对象的命名必须规范。

4.3.2　表和字段的设计

1. 表设计原则

（1）标准化和规范化

数据的标准化有助于消除数据库中的数据冗余。标准化有多种形式，但 Third Normal Form（3NF）通常被认为在性能、扩展性和数据完整性方面达到了最好平衡。简单来说，遵守 3NF 标准的数据库的表设计原则是："One Fact in One Place"即某个表只包括其本身基本的属性，当不是它们本身所具有的属性时需进行分解。表之间的关系通过外键相连接。它具有以下特点：有一组表专门存放通过键连接起来的关联数据。

例如，某个存放客户及其有关订单的 3NF 数据库可能有两个表，即 Customer 和 Order。Order 表不包含订单关联客户的任何信息，但表内会存放一个键值，该键指向 Customer 表中包含该客户信息的那一行。

事实上，为了效率的缘故，对表不进行标准化有时也是必要的。

（2）数据驱动

采用数据驱动而非硬编码的方式，许多策略变更和维护都会方便得多，大大增强了系统的灵活性和扩展性。

例如，假如用户界面要访问外部数据源（文件、XML 文档、其他数据库等），不妨把相应的连接和路径信息存储在用户界面支持表中。还有，如果用户界面执行工作流之类的任务（发送邮件、打印信笺、修改记录状态等），那么产生工作流的数据也可以存放在数据库中。角色权限管理也可以通过数据驱动来完成。事实上，如果过程是数据驱动的，即可把相当大的责任推给用户，由用户来维护自己的工作流过程。

（3）考虑各种变化

在设计数据库时考虑到哪些数据字段将来可能会发生变更。

例如，姓氏就是如此（注意是西方人的姓氏，如女性结婚后从夫姓等）。所以，在建立系统存储客户信息时，在单独的一个数据表中存储姓氏字段，而且还附加起始日和终止日等字段，这样即可跟踪这一数据条目的变化。

2. 字段设计原则

（1）每个表中都应该添加的 3 个有用的字段

- dRecordCreationDate：在 VB 下默认为 Now()，而在 SQL Server 下默认为 GETDATE()。
- sRecordCreator：在 SQL Server 下默认为 NOT NULL DEFAULT USER。
- nRecordVersion：记录的版本标记，有助于准确说明记录中出现 NULL 数据或者丢失数据的原因。

（2）对地址和电话采用多个字段

描述街道地址就短短一行记录是不够的。Address_Line1、Address_Line2 和 Address_Line3 可以提供更大的灵活性。还有，电话号码和邮件地址最好拥有自己的数据表，其间具有自身的类型和标记类别。

（3）使用角色实体定义属于某类别的列

在需要对属于特定类别或者具有特定角色的事物做定义时，可以用角色实体来创建特定的时间关联关系，从而可以实现自我文档化。

例如，用 PERSON 实体和 PERSON_TYPE 实体来描述人员。例如，当 John Smith，Engineer 提升为 John Smith，Director 乃至最后升到 John Smith，CIO 的高位，而所有要做的不过是改变两个表 PERSON 和 PERSON_TYPE 之间关系的键值，同时增加一个日期/时间字段来知道变化是何时发生的。这样，PERSON_TYPE 表就包含了所有 PERSON 的可能类型，如 Associate、Engineer、Director、CIO 或者 CEO 等。还有个替代办法就是改变 PERSON 记录来反映新头衔的变化，不过这样一来在时间上无法跟踪个人所处位置的具体时间。

（4）选择数字类型和文本类型尽量充足

在 SQL 中使用 smallint 和 tinyint 类型要特别注意。假如想查看月销售总额，总额字段类型是 smallint，那么，如果总额超过了$32767 就不能进行计算操作了。

而 ID 类型的文本字段，如客户 ID 或订单号等都应该设置得比一般想象更大。假设客户 ID 为 10 位数长。那应该把数据库表字段的长度设为 12 或者 13 个字符长。但这额外占据的空间却无须将来重构整个数据库即可实现数据库规模的增长。

（5）增加删除标记字段

在表中包含一个“删除标记”字段，这样即可把行标记为删除。在关系数据库中不要单独删除某一行，最好采用清除数据程序而且要仔细维护索引整体性。

4.3.3 选择键和索引

键选择原则如下：

（1）键设计 4 原则

- 为关联字段创建外键。
- 所有的键都必须唯一。
- 避免使用复合键。
- 外键总是关联唯一的键字段。

（2）使用系统生成的主键

设计数据库时采用系统生成的键作为主键，那么实际控制了数据库的索引完整性。这样，数据库和非人工机制就有效地控制了对存储数据中每一行的访问。采用系统生成键作为主键还有一个优点：当拥有一致的键结构时，找到逻辑缺陷很容易。

（3）不要使用用户的键

在确定采用什么字段作为表的键时，可一定要小心用户将要编辑的字段。通常情况下不要选择用户可编辑的字段作为键，即不让主键具有可更新性。

（4）可选键有时可作主键

把可选键进一步用作主键，可以拥有建立强大索引的能力。

4.3.4　索引使用原则

索引是从数据库中获取数据的最高效方式之一。95%的数据库性能问题都可以采用索引技术得到解决。

（1）逻辑主键使用唯一的成组索引，对系统键（作为存储过程）采用唯一的非成组索引，对任何外键列采用非成组索引。考虑数据库的空间有多大，表如何进行访问，还有这些访问是否主要用作读写。

（2）大多数数据库都索引自动创建的主键字段，但是可别忘了索引外键，它们也是经常使用的键，例如运行查询显示主表和所有关联表的某条记录就用得上。

（3）不要索引 memo/note 字段，也不要索引大型字段（有很多字符），这样做会让索引占用太多的存储空间。

（4）不要索引常用的小型表。不要为小型数据表设置任何键，假如它们经常有插入和删除操作就更不要这样做。对这些插入和删除操作的索引维护可能比扫描表空间消耗更多的时间。

4.3.5　数据完整性设计

1. 完整性实现机制

（1）实体完整性：主键。

（2）参照完整性。

- ❑ 父表中删除数据：级联删除、受限删除、置空值。
- ❑ 父表中插入数据：受限插入、递归插入。
- ❑ 父表中更新数据：级联更新、受限更新、置空值。

DBMS 对参照完整性可以有两种方法实现：外键实现机制（约束规则）和触发器实现机制。

（3）用户定义完整性：NOT NULL、CHECK、触发器。

2. 用约束而非商务规则强制数据完整性

采用数据库系统实现数据的完整性。这不但包括通过标准化实现的完整性而且还包括数据的功能性。在写数据时还可以增加触发器来保证数据的正确性。不要依赖于商务层保证数据完整性，它不能保证表之间（外键）的完整性，所以不能强加于其他完整性规则之上。

3. 强制指示完整性

在有害数据进入数据库之前将其剔除。激活数据库系统的指示完整性特性。这样可以保持数据的清洁而能迫使开发人员投入更多的时间处理错误条件。

4. 使用查找控制数据完整性

控制数据完整性的最佳方式就是限制用户的选择。只要有可能都应该提供给用户一个清晰的价值列表供其选择。这样将减少输入代码的错误和误解，同时提供数据的一致性。某些公共数据特别适合查找国家代码、状态代码等。

5. 采用视图

为了在数据库和应用程序代码之间提供另一层抽象，可以为应用程序建立专门的视图，而不必非要应用程序直接访问数据表。这样做还等于在处理数据库变更时提供了更多的自由。

4.3.6 其他设计技巧

（1）避免使用触发器

触发器的功能通常可以用其他方式实现。在调试程序时触发器可能成为干扰。假如确实需要采用触发器，最好集中对它文档化。

（2）使用常用英语（或者其他任何语言）而不要使用编码

在创建下拉菜单、列表、报表时最好按照英语名排序。假如需要编码，可以在编码旁附上用户知道的英语。

（3）保存常用信息

让一个表专门存放一般数据库信息非常有用。在这个表中存放数据库当前版本、最近检查/修复（对 Access）、关联设计文档的名称、客户等信息。这样可以实现一种简单机制跟踪数据库，当客户抱怨他们的数据库没有达到希望的要求而与你联系时，这样做对非客户机/服务器环境特别有用。

（4）包含版本机制

在数据库中引入版本控制机制来确定使用中的数据库的版本。时间一长，用户的需求总是会改变的。最终可能会要求修改数据库结构。把版本信息直接存放到数据库中更为方便。

（5）编制文档

对所有的快捷方式、命名规范、限制和函数都要编制文档。

采用给表、列、触发器等加注释的数据库工具。对开发、支持和跟踪修改非常有用。

对数据库文档化，或者在数据库自身的内部或者单独建立文档。这样，当过了一年多时间后再回过头来做第二个版本，犯错的机会将大大减少。

（6）测试、测试、反复测试

建立或者修订数据库之后，必须用用户新输入的数据测试数据字段。最重要的是，让用户进行测试并且同用户一道保证选择的数据类型满足商业要求。测试需要在把新数据库投入实际服务之前完成。

（7）检查设计

在开发期间检查数据库设计的常用技术是通过其所支持的应用程序原型检查数据库。换句话说，针对每一种最终表达数据的原型应用，保证检查了数据模型并且查看如何取出数据。

4.4　Power Designer 10 简介

PowerDesigner 是 Sybase 公司的 CASE 工具集，使用它可以方便地对管理信息系统进行分析设计，它几乎包括了数据库模型设计的全过程。利用 PowerDesigner 可以制作数据流程图、概念数据模型、物理数据模型，可以生成多种客户端开发工具的应用程序，还可为数据仓库制作结构模型，也能对团队设计模型进行控制。它可与许多流行的数据库设计软件，例如 PowerBuilder、Delphi、VB 等相配合使用，来缩短开发时间和使系统设计更优化。

PowerDesigner 主要包括以下几个功能部分。

（1）DataArchitect

这是一个强大的数据库设计工具，使用 DataArchitect 可利用实体-关系图为一个信息系统创建“概念数据模型”。

创建概念数据模型（Conceptual Data Model，CDM），并且可根据 CDM 产生基于某一特定数据库管理系统（如 Sybase System 11）的物理数据模型（Physical Data Model，PDM）。

还可优化 PDM，产生为特定 DBMS 创建数据库的 SQL 语句并可以文件形式存储，以便在其他时刻运行这些 SQL 语句创建数据库。另外，DataArchitect 还可根据已存在的数据库反向生成 PDM、CDM 及创建数据库的 SQL 脚本。

（2）ProcessAnalyst

这部分用于创建功能模型和数据流图，创建“处理层次关系”。

（3）AppModeler

这部分为客户/服务器应用程序创建应用模型。

（4）ODBC Administrator

这部分用来管理系统的各种数据源。

PowerDesigner 的 4 种模型文件，CDM、PDM 和 OOM 之间的关系如图 4-2 所示。

（1）概念数据模型（CDM）

CDM 表现数据库的全部逻辑结构，与任何软件或数据储藏结构无关。一个概念模型经常包括在物理数据库中仍然不实现的数据对象。它给运行计划或业务活动的数据一个正式表现方式。

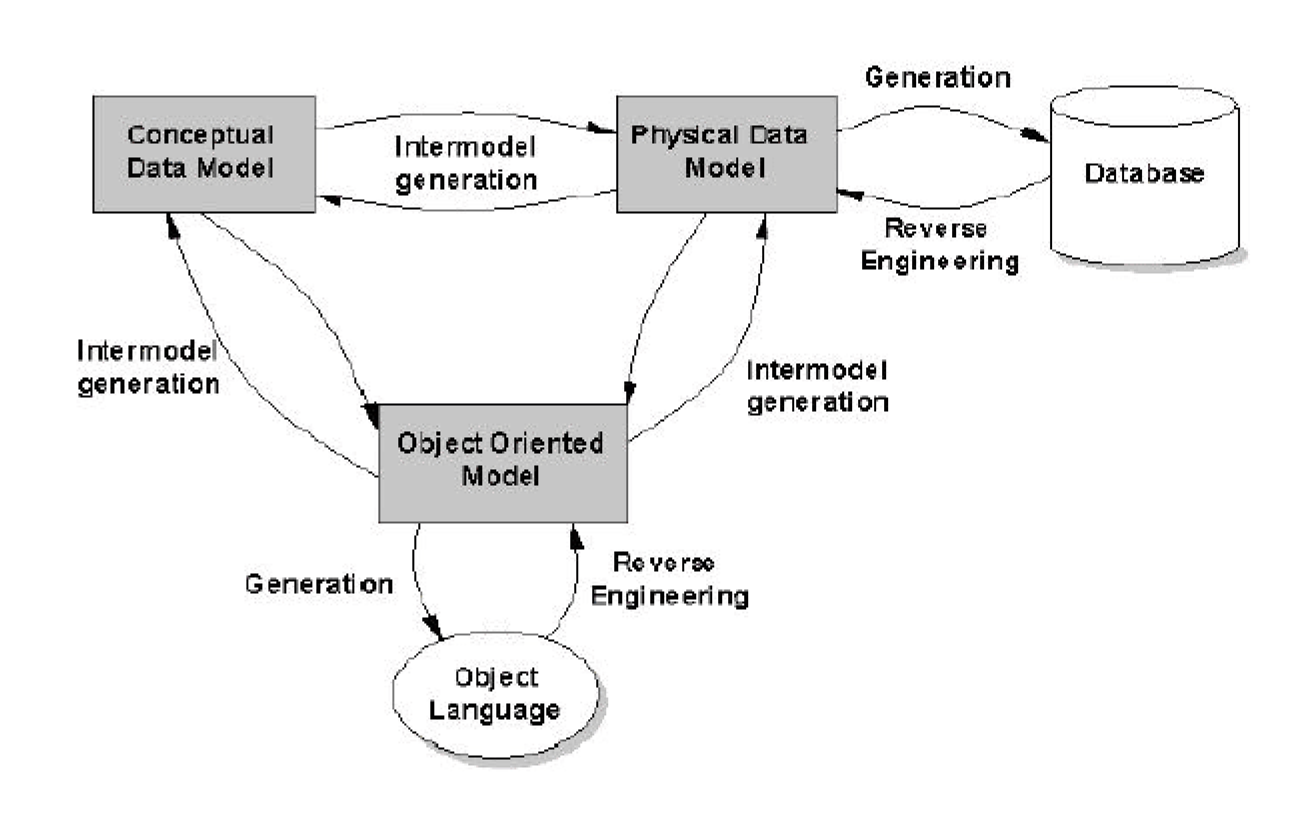


图 4-2　CDM、PDM 和 OOM 之间的关系

（2）物理数据模型（PDM）

PDM 叙述数据库的物理实现，考虑真实的物理实现的细节。

（3）面向对象模型（OOM）

一个 OOM 包含一系列包、类、接口和它们的关系。这些对象一起形成所有的（或部分）一个软件系统的逻辑的设计视图的类结构。一个 OOM 本质上是软件系统的一个静态的概念模型。

使用 PowerDesigner 面向对象模型能为纯粹对象建立一个 OOM，产生 Java 文件或者 PowerBuilder 文件，或能使用一个来自 OOM 的物理数据模型（PDM）对象，来表示关系数据库设计分析。

（4）业务程序模型（BPM）

BPM 描述业务的各种不同内在任务和内在流程，而且客户如何以这些任务和流程互相影响。

BPM 是从业务合伙人的观点来看业务逻辑和规则的概念模型，使用一个图表描述程序、流程、信息和合作协议之间的交互作用。

4.5　Power Designer 10 的使用

1. 使用 Power Designer 环境

（1）树形模型浏览器

对象浏览器可以用分层结构显示工作空间。

（2）输出窗口

输出窗口显示操作的结果。

（3）结果列表

结果列表用于显示生成、覆盖和模型检查结果，以及设计环境的总体信息。

（4）图表窗口

图表窗口用于组织模型中的图表，以图形方式显示模型中各对象之间的关系。

Power Designer 工作环境如图 4-3 所示，其他的窗口与其他的软件大同小异。

2. Power Designer 的操作方式

“关系属性”对话框如图 4-4 所示。

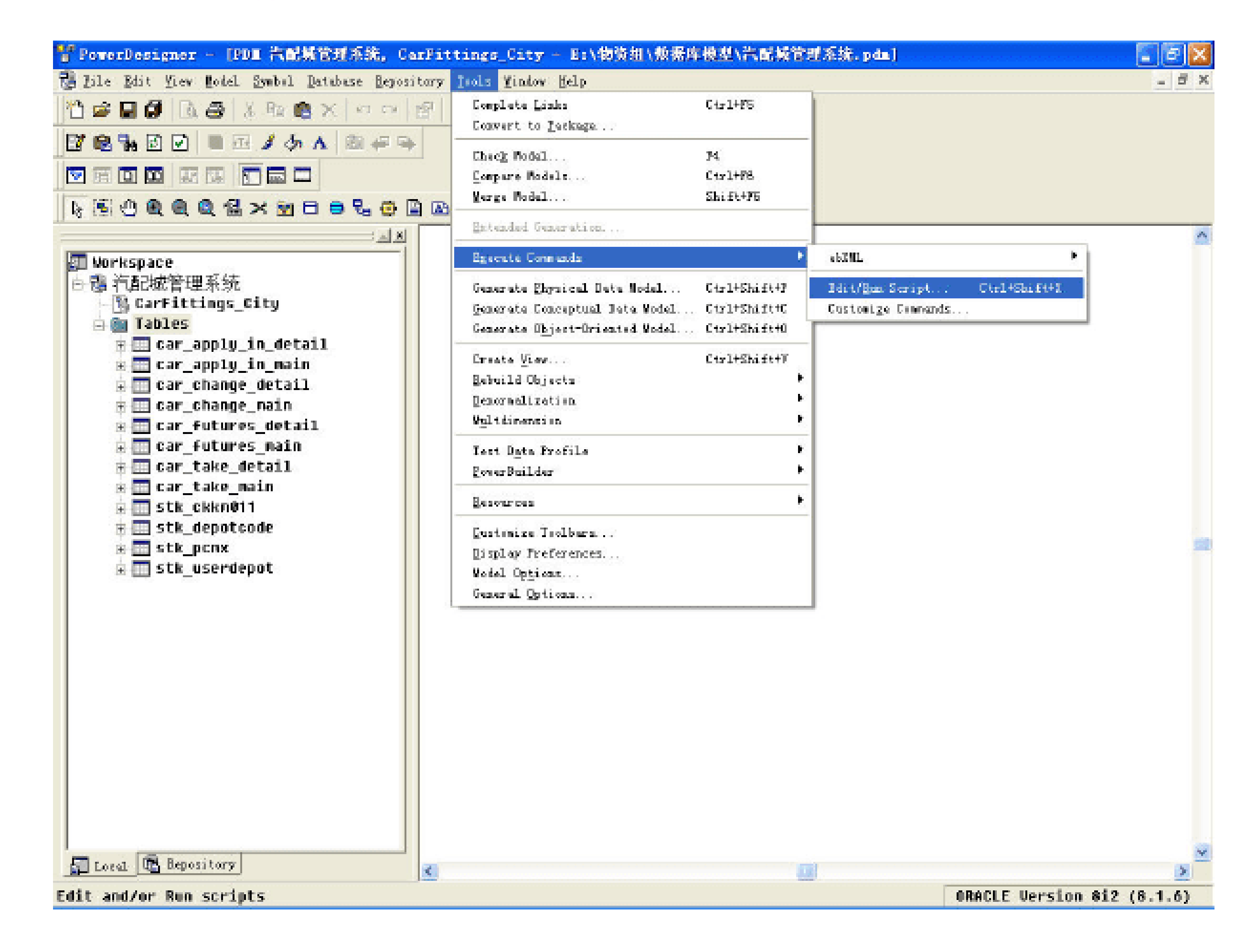


图 4-3　Power Designer 工作环境



图 4-4　“关系属性”对话框

（1）单击工具面板的实体工具。

当光标移动到图表时，变成实体的形状。

（2）在 CDM 图表中单击任何一处。

一个实体符号在单击位置出现。实体名称为 Entity_n，n 是一个创建对象的次序编号。

（3）实体工具仍然是可使用的，因此再一次单击在 CDM 图表中产生另外的一个实体。

现在有 CDM 图表的两个实体。

（4）单击工具面板的关系工具。

实体工具被现在释放，而且关系工具是可使用的。

（5）单击在第一个实体之内而且当继续按住鼠标左键时，拖动鼠标光标到第二个实体。在第二个实体之内放开鼠标左键。这样可以产生关系。

（6）单击鼠标右键，释放关系工具。

一个工具保持可使用直到释放它。释放一个工具，可以选择另外的一个工具或单击鼠标右键。默认情况下，当单击鼠标右键，指针工具被激活。

（7）单击面板的套索工具，套索工具变为可使用。

（8）在第一个实体的上面角落单击，按住鼠标左键不放，拖动鼠标光标拉一个包括两个实体的长方形，放开鼠标左键，实体和关系被选择。

（9）拖动实体到一个新位置，关系跟随实体一起移动。

（10）单击面板的文本工具。文本工具变为可使用。

（11）在关系下面单击，一些文本在被长方形指出的区域中出现。

（12）单击鼠标右键，释放文本工具。

（13）双击文本，一个文本框出现。

（14）在文本框中输入短文本。

（15）单击 OK 按钮，文本在图表中出现。

（16）单击文本框的一个柄，按住鼠标左键，拖动光标到右边直到所有的文本出现，放开鼠标左键，在图表背景上单击，文本框柄消失。

（17）单击面板的指针工具。使用这个工具选择并且删除符号。

（18）在实体符号上单击，选择想删除的对象。

（19）按 Delete 键，出现确认信息对话框，提示如何删除选择。

如果选择删除对象，将删除图标符号并且删除模型中的对象。如果只选择删除符号，将删除图标符号，但是保存模型的对象。

（20）单击 OK 按钮，图表中的实体和联合的关系被移动。对象也从模型删除。

（21）单击剩余的实体，当单击文本时，按 Shift 键，两个对象将被选择。

（22）按 Delete 键，并且在删除信息出现时单击 OK 按钮，剩余的实体和文本被删除。

4.6　正向工程与逆向工程

1．正向工程

从 PDM 能直接产生一个数据库，或产生一个能在数据库管理系统环境中运行的数据库脚本，这是正向工程。

默认是生成与PDM相同类型数据库的脚本，但也支持产生其他类型数据库的创建脚本。

（1）选择 Database→Generate Database 命令，弹出“数据库生成”对话框，如图 4-5 所示。它显示生成参数，默认参数已经被选择。

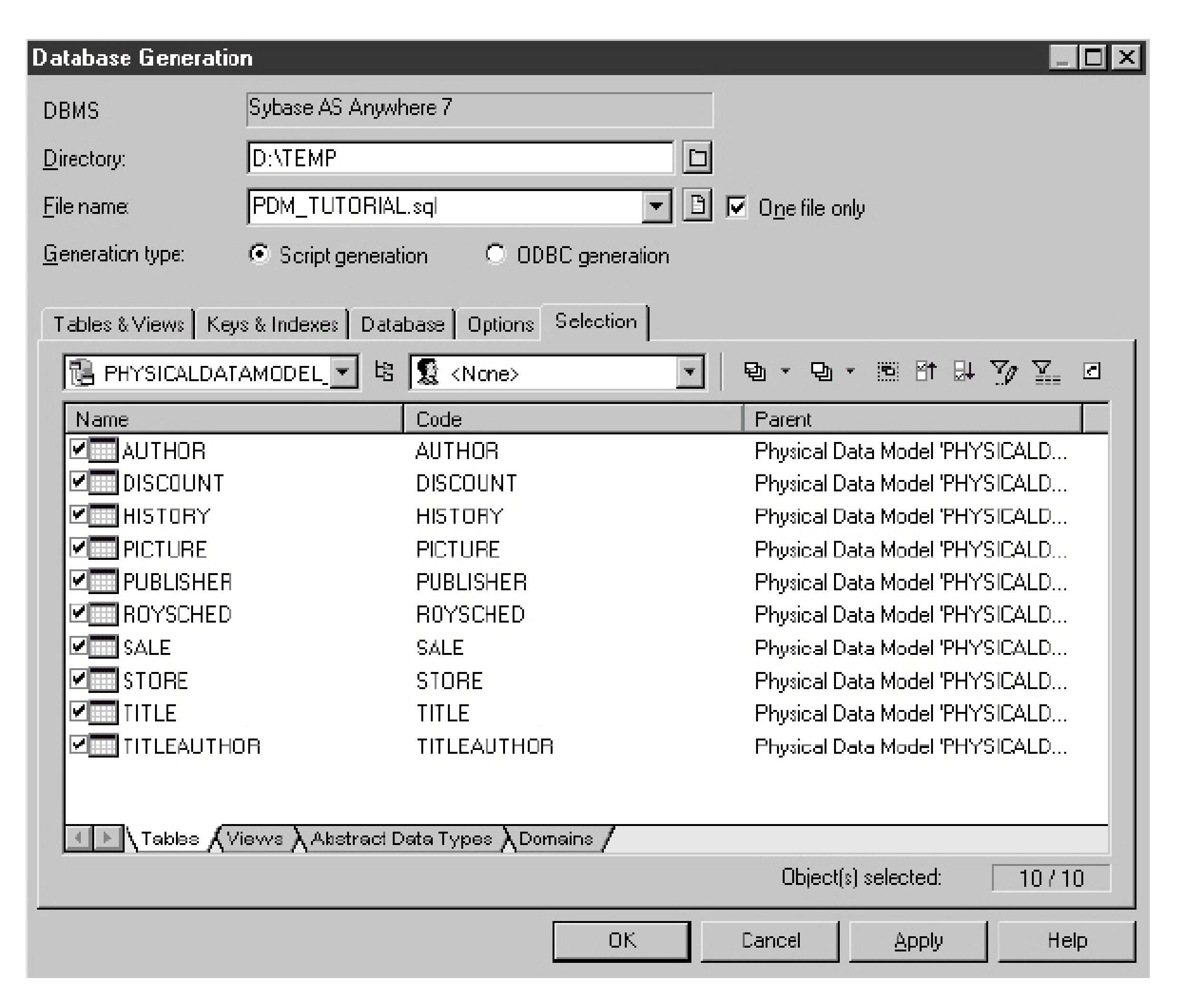


图 4-5　“数据库生成”对话框

（2）在 SQL 的 File name 文本框中输入“PDM_TUTORIAL.Sql”。

（3）在 Directory 文本框中输入一条路径。

（4）选中 Script generation 单选按钮。

（5）选中 One file only 单选按钮。

（6）选择 Selection 选项卡。

（7）选择 Tables 定位键。

表页列出模型中选择可用的所有数据库表。

（8）选中所有选择工具，即选中所有的表复选框。

（9）切换 Views 和 Domains 选项卡选择需要的视图和域。

（10）单击 OK 按钮。

可以生成数据库脚本，如果选择 ODBC 方式，则可以直接连接到数据库，从而直接产生数据库表以及其他数据库对象。

2. 逆向工程

逆向工程已存在的数据库，数据来源可能是从脚本文件或一个开放数据库连接数据来源。当逆向工程使用脚本时，能使用一个单一脚本文件或一些脚本文件。

1）逆向工程数据库对象从一个脚本文件到新的 PDM

逆向工程来自一个脚本文件的数据库对象：

（1）选择 File→Reverse Engineer→Database 命令，弹出“新的物理数据模型”对话框，如图 4-6 所示。

（2）选中 Share：Use the shared DBMS definition 单选按钮。

（3）选择下拉列表框中的一个数据库管理系统。

（4）单击 OK 按钮，弹出“数据库逆向工程”对话框，如图 4-7 所示。

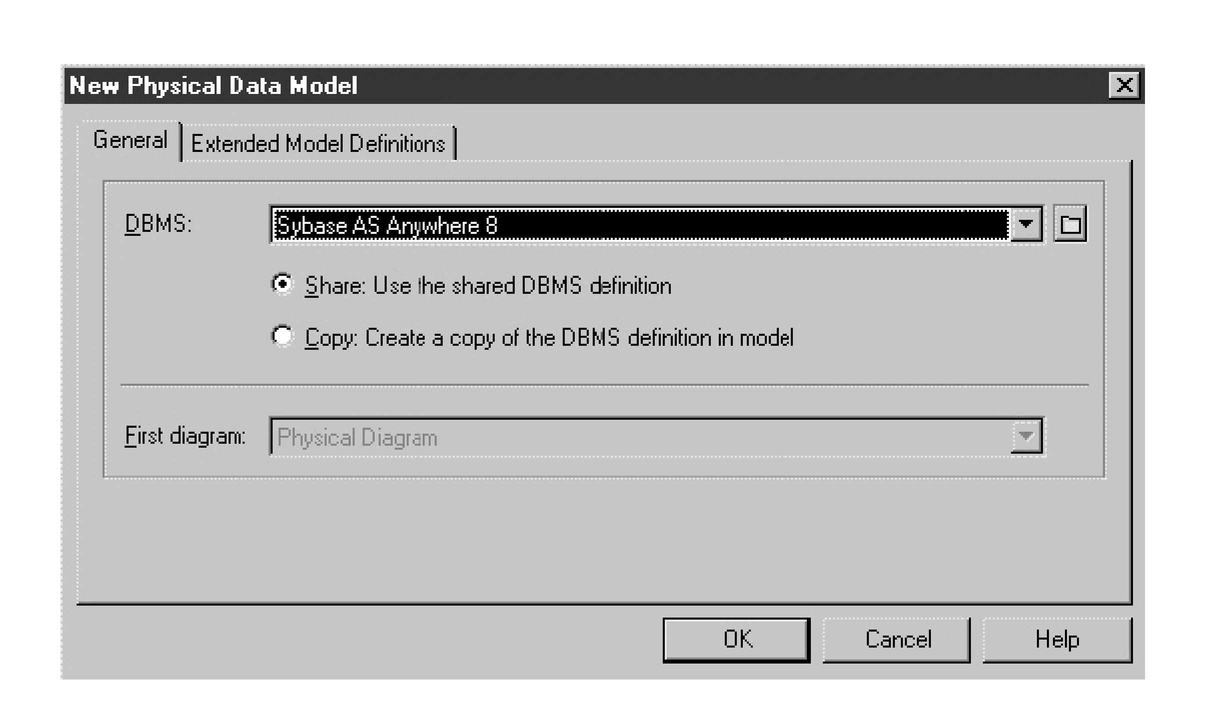


图 4-6　“新的物理数据模型”对话框

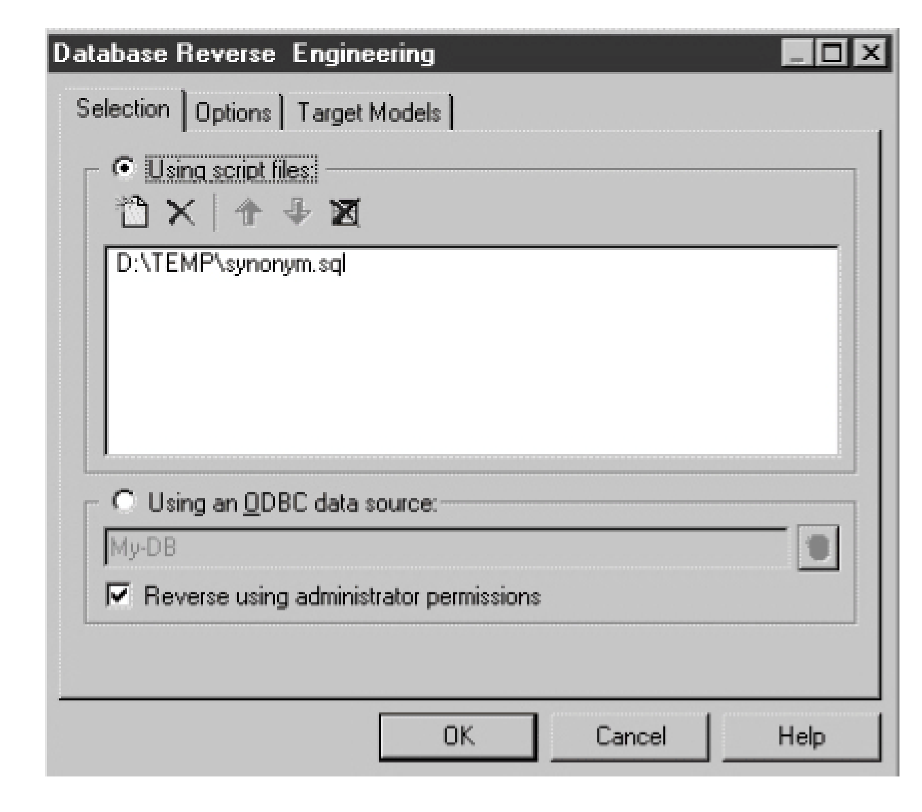


图 4-7　选择脚本文件

（5）选中 Using script files 单选按钮。

（6）浏览适当的目录选择脚本文件。

（7）选择 Options 选项卡。

（8）选中 Create symbols 复选框，如图 4-8 所示。

（9）单击 OK 按钮。

输出窗口的信息指出被指定的文件完全逆向工程。

2）逆向工程一个 ODBC 到新的 PDM

（1）选择 File→Reverse Engineer→Database 命令，弹出“新的物理数据模型”对话框。

（2）选中 Share：Use the shared DBMS definition 单选按钮。

（3）选择下拉列表框中的一个数据库管理系统。

（4）单击 OK 按钮，弹出“数据库逆向工程”对话框，如图 4-9 所示。

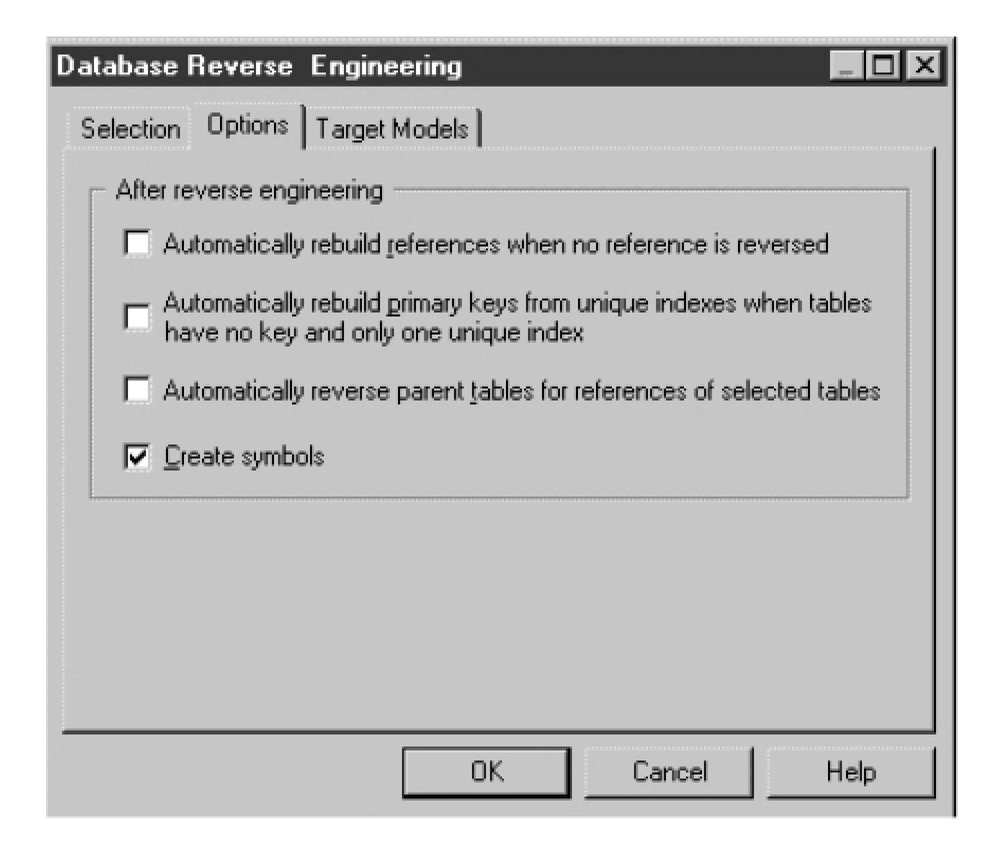



图 4-8　选中 Create symbols 复选框

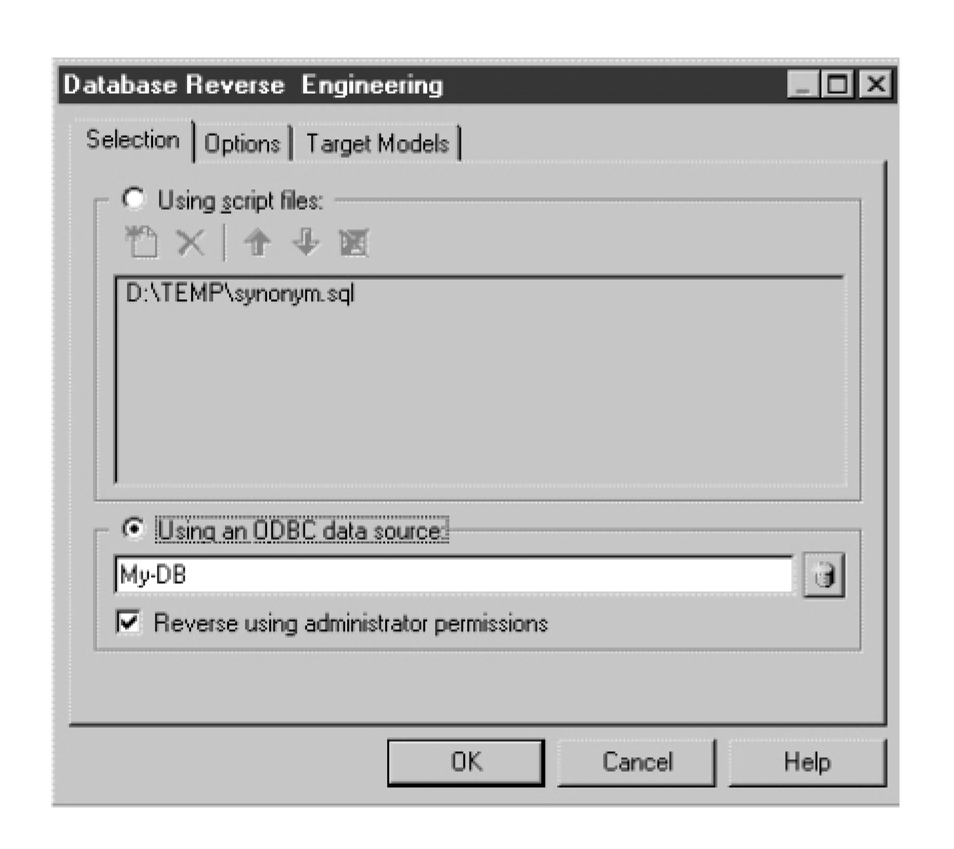


图 4-9　“数据库逆向工程”对话框

（5）选中 Using an ODBC data source 单选按钮，选择一个 ODBC。

（6）选择 Options 选项卡，如图 4-10 所示。

（7）选中 Create symbols 复选框。

（8）单击 OK 按钮，弹出“ODBC 逆向工程”对话框，如图 4-11 所示。

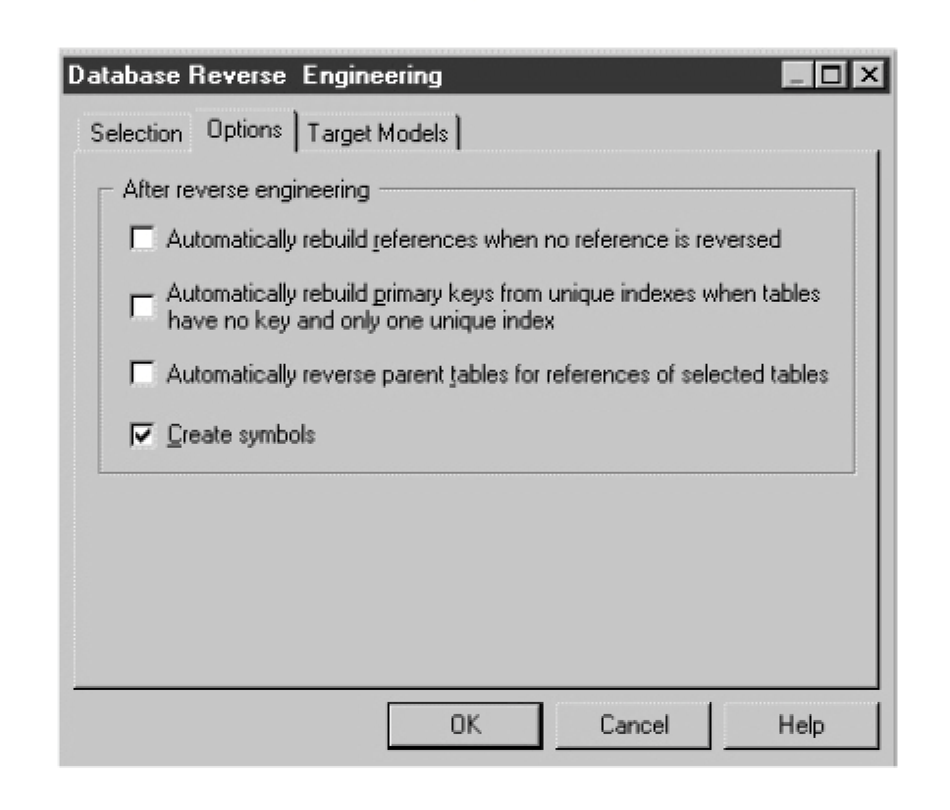


图 4-10　选择 Options 选项卡

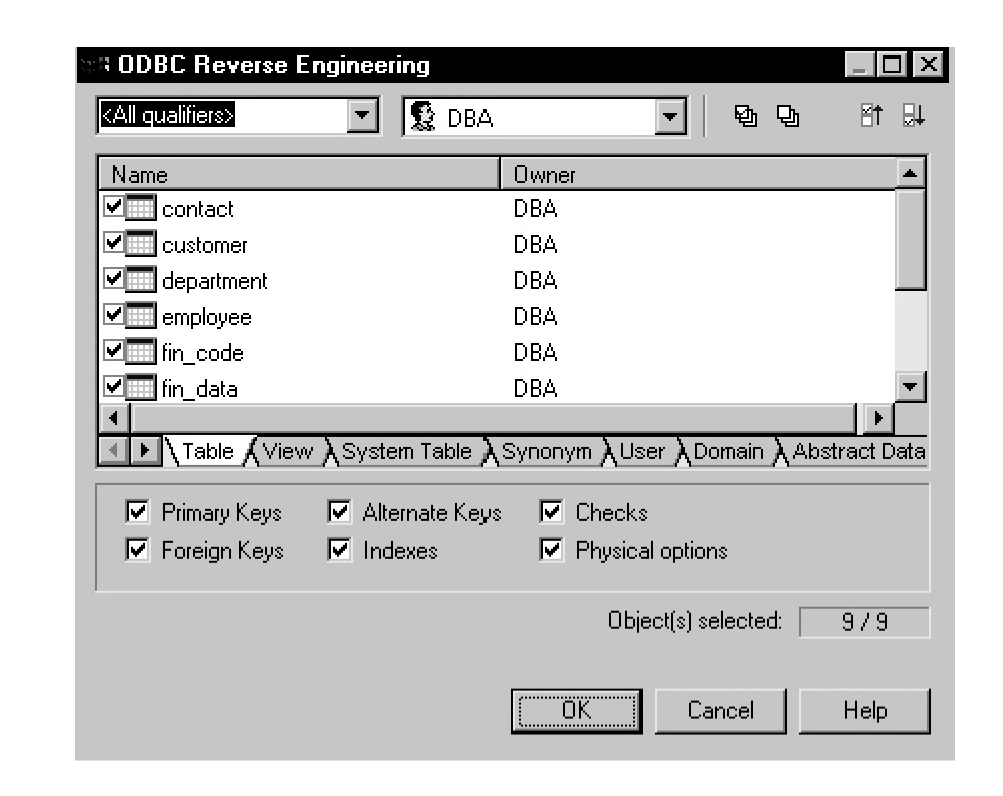


图 4-11　“ODBC 逆向工程”对话框

（9）在对话框的两个下拉列表框中分别选择限定词和拥有者。

（10）单击一个对象类型定位键。

（11）单击 OK 按钮。

输出窗口的信息显示哪些表被转换，而且指出数据库成功逆向工程。

4.7　小　　结

数据库设计是指对于一个给定的应用环境，构造最优的数据库模式，建立数据库及其应用系统，使之能够有效地存储数据，满足各种用户的应用需求。一般把数据库设计过程

分为需求分析、概念结构设计、逻辑结构设计、数据库物理设计、数据库实施、数据库运行与维护 6 个阶段。

数据库的设计范式是数据库设计所需要满足的规范，满足这些规范的数据库是简洁的、结构明晰的。

数据库设计技巧包括需求分析阶段、选择键和索引、数据完整性设计、表和字段的设计等一系列设计技巧。

Power Designer 10 是 Sybase 公司的 CASE 工具集，使用它可以方便地对管理信息系统进行分析设计，它几乎包括了数据库模型设计的全过程，包括正向工程与逆向工程。

第 5 章　存储过程、触发器

存储过程是一些由 MySQL 服务直接存储和执行的定制过程或函数。存储过程的加入把 SQL 语言扩展成了一种程序设计语言，可以利用存储过程把一个客户/服务器体系的数据库应用软件中的部分逻辑保存起来供日后使用。本章将介绍 MySQL 中的存储过程实现细节，并提供一些存储过程应用示例。

触发器是在 Insert、Update 或 Delete 命令之前或之后对 SQL 命令或存储过程的自动调用。例如，可以为 Update 操作测试被修改的数据是否满足特定的条件。

5.1　存储过程和授权表

存储过程是由控制流语句和 SQL 语句书写的过程，这个过程经编译和优化后存储在数据库服务器中，使用时只要调用即可。它不仅可以带有输入参数还可以带有输出参数，存储过程能够通过接收参数向调用者返回结果集，结果集的格式由调用者确定。返回状态者给调用者，并指明调用是成功还是失败，包括针对数据库的操作语句，可以在一个存储过程中调用另一个存储过程。

使用存储过程的主要优点有以下几个方面。

❑ 效率高

在进行数据库操作时，经常会出现在 JSP 程序和数据库服务器之间来回传输大量数据的情况。随着用户数的增加，SQL 请求也就不断地增加，使网络很快就成为运行的瓶颈。使用存储过程可使运行性能得到显著的改进，因为存储过程的一次调用，即调用了在服务器中执行的多个 SQL 语句，从而减少了网络的拥挤。

❑ 可重用性强

一个存储过程只需编写一次，即可用于各种地方——SQL 脚本、数据库触发器和客户机应用程序。

❑ 提高数据的安全性和完整性

由于存储过程在数据库服务器执行，这样就大大提高了数据库的安全性。可以根据具体情况为不同的数据和数据访问操作设置不同严格程度的安全检查规则。通过存储过程可以使相关的动作在一起发生，从而可以维护数据库的完整性。

如图 5-1 所示是把执行逻辑放入应用程序中的界面。

如图 5-2 所示是把执行逻辑放入存储过程中的界面。

图 5-3 显示了存储过程的处理过程。左边部分是要调用存储过程的应用程序，中间部分是数据库服务器，右边部分是数据库和目录。具体执行步骤描述如下：



图 5-1　执行逻辑放入应用程序中



图 5-2　执行逻辑放入存储过程中

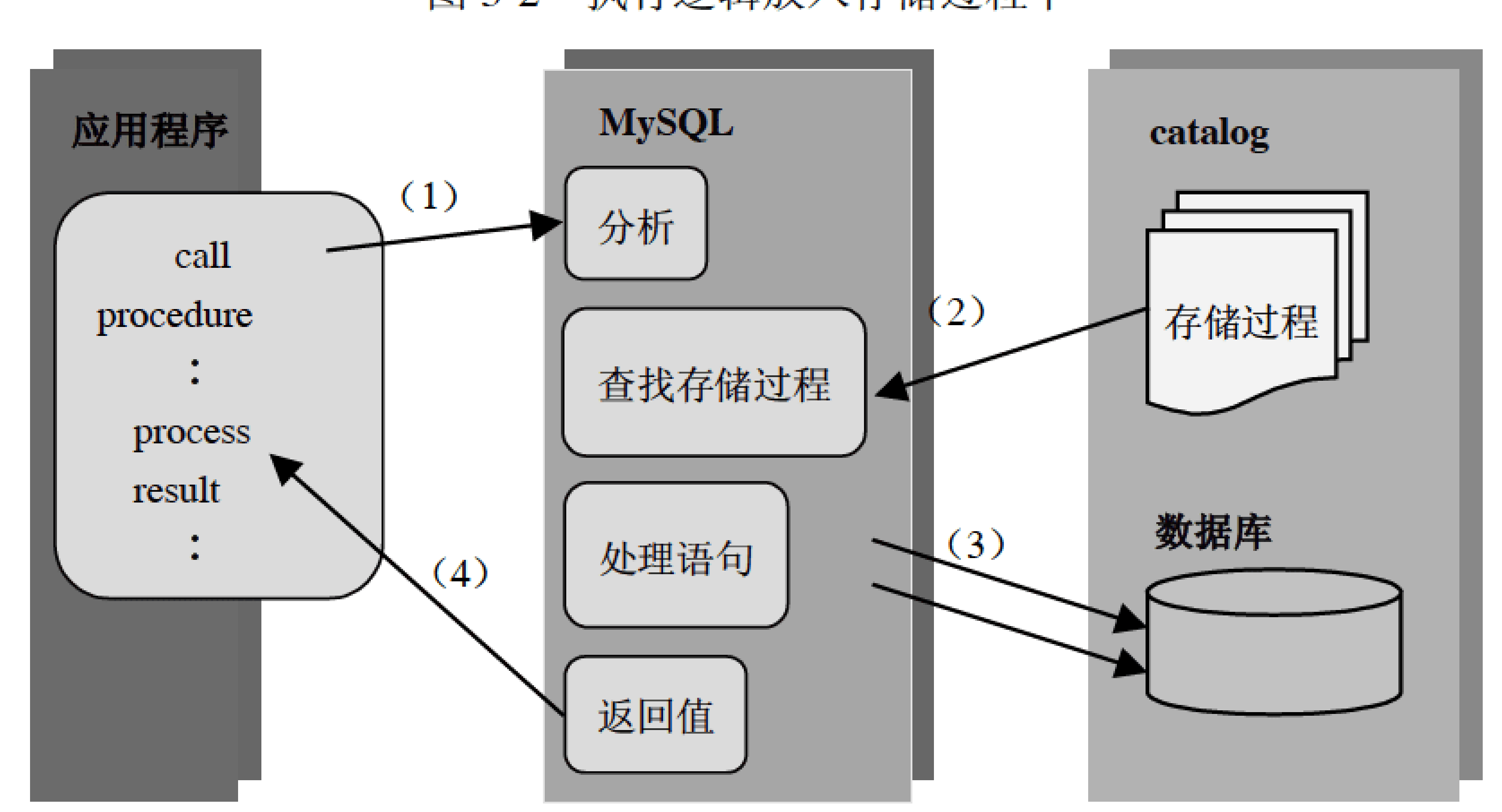


图 5-3　存储过程的处理步骤

（1）应用程序中开始调用存储过程。

（2）数据库服务器接收调用请求，并在目录中查找匹配的存储过程。

（3）执行存储过程。

（4）一旦存储过程执行完毕，返回执行的结果。

存储过程在 MySQL 数据库中需要 proc 数据表。该数据表是在 MySQL 安装时自动创建的。这个数据表的各个列中存放着存储过程所属的数据库名、存储过程的名称和类型、存储过程的参数、存储过程的实际代码以及其他属性。

【特别提示】如果从早期的 MySQL 版本升级到 MySQL 5.1，要更新授权表以确保 mysql.proc 表的存在。当更新到新版本 MySQL 时，要想确保授权表最新，应当运行 mysql_fix_privilege_tables 脚本来更新授权表。如果从 MySQL 4.1 或更早版本升级，授权表升级过程为 CREATE VIEW 和 SHOW VIEW 权限增加了视图相关的列。这些权限位于全局数据库级。在这种情况下，MySQL 5.1 版本的 MySQL_fix_privilege_tables 将 user 表中的 Create_priv 值复制到 Create_view_priv 和 Show_view_priv 列。

在 MySQL 5.1 数据库中，授权系统有如下考虑：

（1）利用 CREATE ROUTINE 权限来创建存储子程序。

（2）利用 ALTER ROUTINE 权限来修改存储子程序。

（3）利用 EXECUTE 权限来执行子程序。

5.2　存储过程的语法

有两种类型的存储过程，一种是过程，另一种是函数。每一个存储过程至少由 3 个部分组成：名称、参数列表和存储体。下面对这两种存储过程进行一下总结，如表 5-1 所示。

表 5-1　过程与函数的区别

项　目	过　程	函　数
调用情况	通过 CALL 命令调用	可以嵌入在所有的 SQL 命令中
参数	可以使用值参数和引用参数（IN、OUT、INOUT）	只能使用值参数，不能使用 IN 等关键字
返回值	返回一个或多个 SELECT 结果	用 RETURN 命令返回一个值
代码中的可用命令	SELECT、INSERT、UPDATE、DELETE、CREATE TABLE 等 SQL 命令	不能使用访问数据表的 SQL 命令
调用其他过程和函数	可以调用其他的过程和函数	只能调用函数，不能调用过程

5.2.1　基本语法规则

存储过程的代码主要由一些 SQL 语言命令构成。下面是存储过程的一些基本语法规则。

- BEGIN-END。由多条 SQL 命令构成的存储过程的代码都必须以 BEGIN 开始，以 END 结束。语法规则如下：

```
BEGIN
   statement_list
 END
```

- 换行符。换行符在存储过程代码中的语义效果与空格字符相同。这就是说把 IF-THEN-ELSE-END-IF 结构连续写在同一行上或分开写在多个行上都是可以的。

- 变量。供存储过程内部使用的局部变量和局部参数不加“@”前缀。在存储过程内允许使用普通的 SQL 变量，但它们必须加“@”前缀。
- 字母大小写情况。存储过程在定义和调用时均不区分字母大小写情况。
- 注释。注释以两个连字符（--）开始并一直延续到这一行的末尾。
- 分号。同一个存储过程可以包含任意多条 SQL 命令。这些命令必须用分号隔开，包括条件和循环的控制结构也必须用分号结束。

5.2.2　条件

1. IF 语句

IF 语句用来定义有条件执行的某些语句。其中 ELSE 子句是可选的。IF 语句的 SQL 语法如下所示。

```
IF condition THEN
  statement_list;
[ELSE IF condition THEN
  statement_list;]
[ELSE
  statement_list;]
END IF;
```

下面是一个 IF 条件的例子。

例 5-1　IF 语句的使用。

```
CREATE  PROCEDURE  if_test
    (IN  p1  INT,
    IN   p2  INT,
    OUT  P3  INT)
  BEGIN
      IF  p1 > p2  THEN
        SET  p3 = 10;
      ELSEIF  p1 = p2  THEN
        SET  p3 = 20;
      ELSE
        SET  p3 = 30;
      ELSE  IF;
  END
```

2. CASE 语句

CASE 语句是 IF 语句的一种语法变体，特别适合用在需要根据同一表达式的不同取值来决定将执行哪一个分支的场合。下面是 CASE 语句的语法。

```
CASE expression
WHEN value1 THEN
  statement_list;
```

```
[WHEN value2 THEN
statement_list;]
  [ELSE
statement_list;]
  END CASE;
```

5.2.3　循环

1. REPEAT-UNTIL 循环

在这种循环结构中，关键字 REPEAT 和 UNTIL 之间的语句将一直循环执行到给定条件第一次得到满足为止。由于对条件表达式的求值发生在每次循环的末尾，所以整个循环至少会执行一次。

这种循环可以有一个可选的标号，此时必须在整个循环语句的末尾也写出同样的标号。给一个循环语句加上标号的目的一般是为了使用 LEAVE 命令提前退出整个循环，或者是为了使用 ITERATE 命令把循环体中的命令再执行一遍。

```
[loopname:] REPEAT
  statement_list;
UNTIL condition
END REPEAT [loopname];
```

下面是一个 REPEAT-UNTIL 循环的例子。

例 5-2　REPEAT-UNTIL 循环语句的使用。

```
CREATE PROCEDURE ru_test (k INT)
BEGIN
    SET @year = 0;
    REPEAT SET @year = @year+1;
    UNTIL  @year>k  END REPEAT;
END
```

2. WHILE 循环

在这种循环结构中，关键字 DO 和 END WHILE 之间的语句将一直循环执行到给定条件第一次没有得到满足为止。因为对条件表达式的求值发生在每次循环的开始，所以如果给定条件在第一次求值时就没有得到满足，整个循环将一次也不执行。如果打算在一个 WHILE 循环中使用 LEAVE 和/或 ITERATE 命令，还必须给这个循环加上一个标号。

```
[loopname:] WHILE condition DO
  statement_list;
END WHILE [loopname];
```

下面是一个 WHILE 循环的例子。

例 5-3　WHILE 循环语句的使用。

```
CREATE PROCEDURE test()
```

```
BEGIN
    DECLARE month INT DEFAULT 12;
    WHILE month>0 DO;
      SET month = month - 1;
    END WHILE;
END
```

3. LOOP 循环

在这种循环结构中，关键字 LOOP 和 END LOOP 之间的语句将一直循环执行到遇见一条 LEAVE loopname 命令并因此而退出整个循环为止。LOOP 循环的语法不要求必须给它们加上一个标号，但在实际运用中它们几乎总是有标号的。

```
Loopname: LOOP
  statement_list;
END LOOP loopname;
```

下面是一个 LOOP 循环例子。

例 5-4　LOOP 循环语句的使用。

```
CREATE PROCEDURE test(month INT)
BEGIN
  myloop: LOOP
    SET month = month+1;
    IF month<=12 THEN ITERATE myloop; END IF;
    LEAVE myloop;
  END LOOP myloop;
  SET @y = month;
END
```

4. LEAVE 和 ITERATE 语句

LEAVE loopname 命令将使用程序代码的执行流程跳出一个循环。LEAVE 命令还可以用来提前退出 BEGIN-END 语句块。

ITERATE loopname 命令的效果是把循环体中的命令再执行一遍。ITERATE 命令不能像 LEAVE 命令那样在 BEGIN-END 语句块中使用。下面举一个例子。

例 5-5　LEAVE 和 ITERATE 语句的使用。

```
CREATE PROCEDURE test(day INT)
BEGIN
  myloop: LOOP
    SET day = day + 1;
    IF day<25 THEN ITERATE myloop; END IF;
    LEAVE myloop;
  END LOOP myloop;
  SET @y = day;
END
```

5.2.4　调用存储过程

过程必须用 CALL 命令来调用。可以返回一个数据表作为过程的调用结果。

```
CALL spname([parameterlist])
```

例如，GetCode()的行为就像是一条 SELECT 命令：

```
CALL getCode(1)
```

CALL 语句可以用声明为 OUT 或 INOUT 参数为它的调用者传递值。它也可以“返回”受影响的行数，客户端程序可以在 SQL 级别通过调用 ROW_COUNT()函数获得此数。如果过程有引用参数（OUT 或 INOUT），它的返回结果将只能通过在调用时给它传递一个 SQL 变量的方法来使用，如下所示：

```
CALL test(10,@result)
SELECT @result
```

5.2.5　参数和返回值

过程必须使用 CREATE PROCEDURE 命令来创建，它们可以有、也可以没有参数表。

```
CREATE PROCEDURE name ([parameterlist])
[options] sqlcode
```

如果有一个以上参数，它们必须用逗号隔开。每个参数都要使用如下所示的语法来定义：

```
[IN or OUT or INOUT] parametername type
```

关键字 IN、OUT 和 INOUT 用来区分有关参数的用途是仅限于输入数据、仅限于输出数据还是输入输出数据均可。type 为任何有效的 MySQL 数据类型。

参数可以是 MySQL 所支持的任何一种数据类型，但无法为它们提供有关数据类型的附加属性（如 NULL 或 NOT NULL），这与为某个数据表定义一个数据列时的情况是不同的。就目前而言，MySQL 在传递参数时是不会进行类型检查的。

存储过程本身没有单个的返回值。不过在过程中可以使用普通的 SELECT 命令，甚至可以连续使用多条 SELECT 命令，而这将使过程返回多个结果数据表。

5.2.6　存储过程的管理

1. 创建存储过程和函数

新的存储过程要用 CREATE FUNCTION 或 CREATE PROCEDURE 命令来创建。但必须具备 Create Routine 权限才能执行这两条命令。下面列出两个命令的语法：

```
CREATE PROCEDURE sp_name ([parameterlist])
[options] sqlcode
```

```
CREATE FUNCTION sp_name ([parameterlist])
RETURNS type
[options] sqlcode
```

过程和函数可以有一样的名字。常见选项介绍如下：

- LANGUAGE SQL。SQL 是 LANGUAGE 选项的默认值，也是仅有的一个可选值。
- [NOT]DETERMINISTIC。假如存储过程针对同样的参数返回的结果总是一样的，就说它是 DETERMINISTIC（可确定的）。假如存储过程的返回结果要取决于数据表，它就是 NOT DETERMINISTIC（不可确定的）。默认情况是 NOT DETERMINISTIC。
- SQL SECURITY[DEFINER|INVOKER]。SQL SECURITY 选项负责设置存储过程在执行时的访问权限。
- COMMENT'string'。注释内容'string'将随存储过程一同存储在 mysql.roc 数据库表中。
- 其他选项。

下面在 MySQL 5.1 中创建一个存储过程和一个函数。

test 表结构如图 5-4 所示。

```
mysql> use test
Database changed
mysql> select * from test;
+------+------+
| code | year |
+------+------+
|    1 |   20 |
|    2 |   20 |
|    3 |   30 |
|    4 |   32 |
+------+------+
4 rows in set (0.00 sec)

mysql>
```

图 5-4　test 表结构

例 5-6　创建存储过程。

```
CREATE PROCEDURE delete_test (IN p INT)
BEGIN
  DELETE
  FROM test
  WHERE year=p;
END;
```

例 5-7　创建函数。

```
CREATE FUNCTION number_of_test()
  RETURNS INT
BEGIN
  RETURN (SELECT COUNT(*) FROM test);
END
```

执行函数：

```
SELECT numner_of_test()
```

2. 修改存储过程和函数

具有 Alter Routine 权限的用户可以用 ALTER 命令来修改存储过程的名称及某些选项。这条命令的语法如下：

```
ALTER PROCEDURE/FUNCTION sp_name
  [NAME newname]
  [SQL SECURITY DEFINER/INVOKER]
  [COMMENT 'newcomment']
```

3. 删除存储过程

删除一个现有的存储过程或函数，可以用以下语法的命令：

```
DROP PROCEDURE [IF EXISTS] name
DROP FUNCTION [IF EXISTS] name
```

其中可选关键字 IF EXISTS 的作用是：用户试图删除的存储过程即使不存在，存储过程删除命令也不会出现一个错误提示。只有具有 Alter Routine 权限的用户才能执行该命令。存储过程或函数的创建用户将自动获取删除存储或函数这一权限。

4. 查看存储过程和函数的代码

如果知道一个存储过程的名称，就可以查看它的代码，具体做法是执行下面的命令：

```
SHOW CREATE PROCEDURE [IF EXISTS] name
SHOW CREATE FUNCTION [IF EXISTS] name
```

5. 查看现有的存储过程和函数

查看现有的存储过程和函数命令语法如下：

```
SHOW PROCEDURE STATUS [LIKE'pattern']
SHOW FUNCTION STATUS [LIKE 'pattern']
```

这个语句是 MySQL 的扩展。它返回子程序的特征，如数据库、名字、类型、创建者及创建和修改日期。其中 LIKE 'pattern'表达式让这两条命令只列出与给定搜索模板相匹配的过程或函数。如果没有指定样式，根据使用的语句，所有的存储过程或函数的信息都被列出。

5.2.7 BEGIN-END 复合语句

超过一条 SQL 命令构成的过程或函数必须以关键字 BEGIN 开头、以关键字 END 结束。另外，在代码内有时也需要使用 BEGIN-END 结构，例如在 IF 条件语句或循环语句中声明局部变量、条件、出错处理或游标等。

在 BEGIN-END 语句块的内部，有关语句或命令要按以下顺序写出：

```
BEGIN
  DECLARE  variables;
  DECLARE  cursors;
  DECLARE  conditions;
  DECLARE  handler;
```

```
  Other SQL commands;
END;
```

复合语句可以被标记。在关键字 BEGIN 的前面可以加上一个可选的标号，如 *labe_name*，那么必须在与之对应的 END 关键字后面也要写出这个标号 *labe_name*。如果两者都存在，它们必须是一样的。

5.2.8　存储过程的变量

1. DECLARE（声明）

DECLARE 仅被用在 BEGIN-END 复合语句中，并且必须在复合语句的开头，在任何其他语句之前。

游标必须在声明处理程序之前被声明，并且变量和条件必须在声明游标或处理程序之前被声明。

下面是变量声明的语法：

```
DECLARE varname1,varname2,… type [DEFAULT value];
```

这个语句被用来声明局部变量。要给变量提供一个默认值，需要包含一个 DEFAULT 选项。值可以被指定为一个表达式，不需要为一个常数。如果没有 DEFAULT 选项，初始值为 NULL。

局部变量的作用范围在它被声明的 BEGIN-END 块内。它可以被用在嵌套的块中，除了那些用相同名字声明变量的块。

2. 变量 SET 语句

在存储程序中的 SET 语句是一般 SET 语句的扩展版本。被参考变量可能是子程序内声明的变量，或者是全局服务器变量。

下面是变量 SET 语句的语法：

```
SET var_name1 = value1,var_name2 = value2 …
```

在存储程序中的 SET 语句作为预先存在的 SET 语法的一部分来实现。其中不同的变量类型（局部声明变量及全局和集体变量）可以被混合起来。这也允许把局部变量和一些只对系统变量有意义的选项合并起来。

3. SELECT…INTO 语句

SELECT…INTO 语句的语法如下：

```
SELECT col_name1,col_name2,…INTO var_name1,var_name2,var_name3,… table_
expr
```

这个 SELECT 语法把选定的列直接存储到变量。因此，只有单一的行可以被取回。

```
SELECT id,data INTO x,y FROM test.t1 LIMIT 1;
```

注意，用户变量名在 MySQL 5.1 中是对大小写不敏感的。

【特别提示】SQL 变量名不能和列名一样。如果 SELECT…INTO 这样的 SQL 语句包含一个对列的参考，并包含一个与列相同名字的局部变量，MySQL 当前把参考解释为一个变量的名字。例如，在下面的语句中，yname 被解释为到 yname variable 的参考而不是到 yname column 的。

例 5-8　SELECT…INTO 语句的使用。

```
CREATE PROCEDURE test (y VARCHAR(10))
 BEGIN
  DECLARE yname VARCHAR(10) DEFAULT 'wzyou';
  DECLARE xname VARCHAR(10);
  DECLARE yid INT;

  SELECT yname,id INTO xname,yid
    FROM table_test WHERE yname = yname;
  SELECT xname;
 END;
```

当这个程序被调用时，无论 table_test.yname 列的值是什么，变量 xname 将返回值 "wzyou"。

5.2.9　游标

存储程序和函数内支持简单的游标。游标向用户提供了遍历数据表中的全部数据记录的可能性。游标实际上是一种能从包括多条数据记录的结果集中每次提取一条记录的机制。游标必须在声明处理程序之前被声明，并且变量和条件必须在声明游标或处理程序之前被声明。

一般来说，应用程序设计语言不能处理数据集合，如 C 语言。因此 SQL 语句访问数据库所返回的结果就不能被程序设计语言所接收。为实现对结果集的逐条处理，MySQL 通过游标来解决这个问题。

例如，通过下列语句查询 employee 表中所有女职员的姓名和部门编号。

```
select  cname,dept  from  employee  where ssex='女'
```

此时得到的查询结果是 employee 表中满足查询条件的结果集，如图 5-5 所示，共有 6 个数据行。若使用游标，打开游标从查询结果集中检索处理一行数据，图 5-5 说明了游标如何充当指针作用来返回一行数据。



	CNAME	DEPT
1	李江云	1187
2	陈钧宪	1132
3	李定美	1144
4	贺丽萍	1144
5	陈蕾	1132
6	张洁	1133

图 5-5　使用游标查询结果

在图 5-5 中，游标此时所在行是“李定美”职员的记录，也可以在结果集上滚动游标，返回结果集中的任一行。只要不关闭，游标就可以一直检索下去，一旦游标关闭，检索就不能再继续下去。

游标的使用都经过声明游标、打开游标、从游标中提取信息和关闭游标 4 个步骤。

1．声明游标

在使用游标检索数据之前必须先创建游标。MySQL 使用 DECLARE CURSOR 语句创建游标。

声明游标语句如下：

```
DECLARE cursor_name CURSOR FOR select …
```

这个语句声明一个游标。也可以在子程序中定义多个游标，但是一个块中的每一个游标必须有唯一的名字。

SELECT 语句不能有 INTO 子句。

当游标被成功创建后，游标名称成为该游标的标识。如果在以后的存储过程、触发器中使用游标，必须指定该游标的名称。

声明游标 employee_cursor，其指向的结果集为 employee 表中所有职员的姓名、所在的部门和年龄信息，在结果集中以年龄的升序排序，如例 5-9 所示。

例 5-9　声明游标。

```
DECLARE employee_cursor CURSOR
FOR
SELECT cname,dept,age FROM employee
ORDER BY age;
```

2．OPEN 语句

游标 OPEN 语句如下：

```
OPEN cursor_name
```

这个语句打开先前声明的游标。

3．游标 FETCH 语句

游标 FETCH 语句如下：

```
FETCH cursor_name INTO var1,var2,…
```

这个语句用指定的打开游标读取下一行（如果有下一行），并且前进游标指针。数据字段的第一个字段将被读入变量 *var1*，第二个字段将被读入变量 *var2*，依此类推。

4．游标 CLOSE 语句

游标 CLOSE 语句如下：

```
CLOSE cursor_name
```

这个语句关闭先前打开的游标。如果未被明确地关闭，游标在它被声明的复合语句的

末尾被关闭。

【特别提示】① 游标只能用来读取数据，不能用来修改数据。② 游标只能前进。③ 在使用某个游标读取数据的同时，这个游标所涉及的数据表都不允许发生任何变化。

图 5-6 显示了执行某种 SQL 语句之后游标所在的位置。

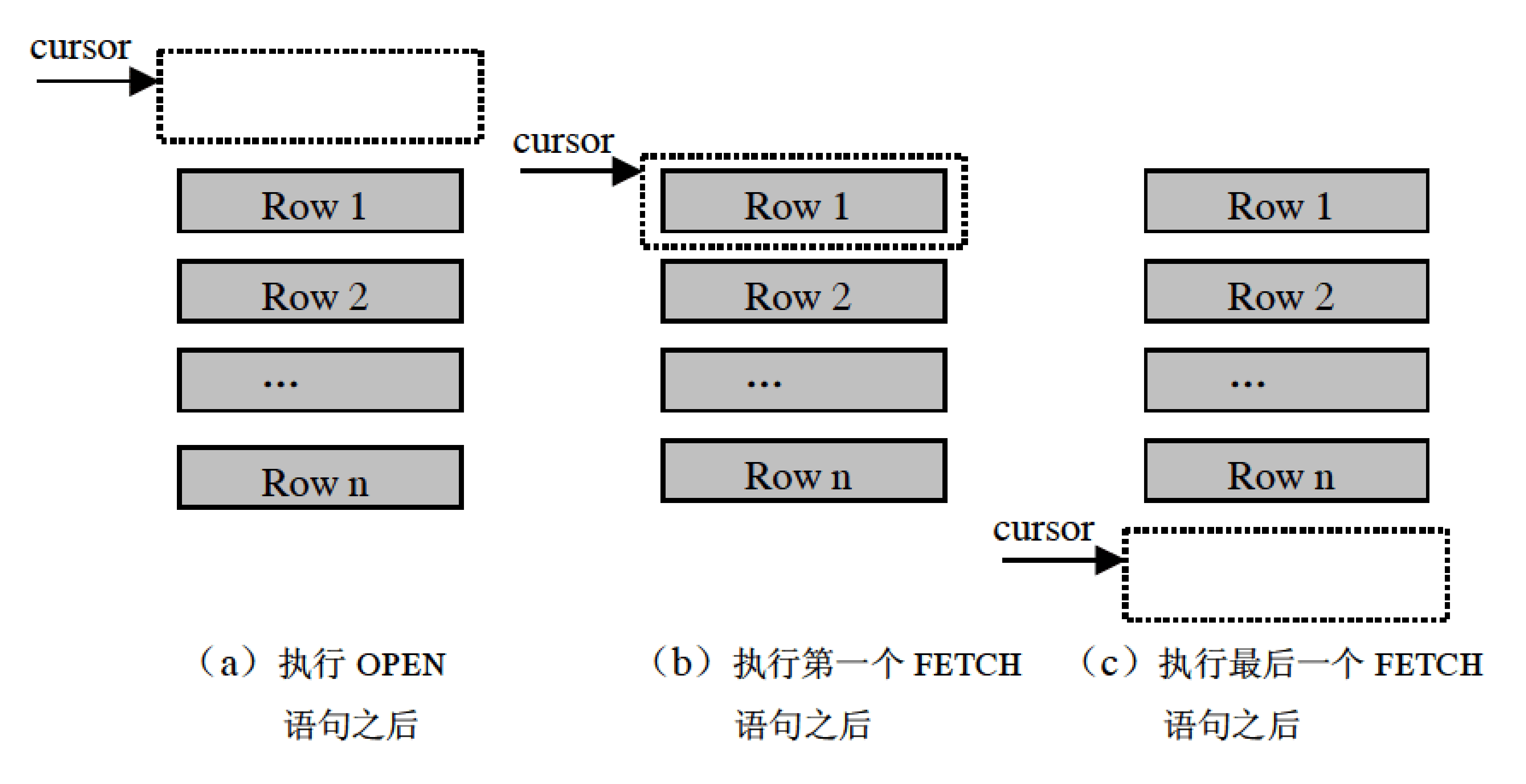


图 5-6　执行特定 SQL 语句后游标所处的位置

游标举例如下。

例 5-10　游标的使用。

```
CREATE PROCEDURE cur_test()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE a,b,c INT;
  DECLARE cur1 CURSOR FOR SELECT code,year FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT month FROM test.t2;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;

  OPEN cur1;  --打开游标
  OPEN cur2;  --打开游标

  REPEAT
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF NOT done THEN
       IF b < c THEN
          INSERT INTO test.t3 VALUES (a,b);
       ELSE
          INSERT INTO test.t3 VALUES (a,c);
       END IF;
    END IF;
  UNTIL done END REPEAT;

    CLOSE cur1;  --关闭游标
```

```
 CLOSE cur2;  --关闭游标
END
```

5.2.10　存储过程应用示例

下面是 MySQL 存储过程和 ORACLE 存储过程应用示例，比较一下两个数据库存储过程的异同点。

MySQL 存储过程应用示例如下。

例 5-11　MySQL 存储过程应用示例。

```
CREATE PROCEDURE total_parents
    (IN  p_playerno  INT,
    INOUT  num  INT)
BEGIN
    DECLARE  v_father, v_mother  INT;
    SET  v_father = (SELECT  father_playerno
                    FROM   players_parents
                    WHERE  playerno = p_playerno);
      SET  v_mother = (SELECT  mother_playerno
                    FROM    players_parents
                    WHERE   playerno = p_playerno);
      IF  v_father  IS  NOT  NULL  THEN
          CALL  total_parents(v_father,num);
          SET  num = num + 1;
      END IF;
      IF  v_mother  IS  NOT  NULL  THEN
          CALL  total_parents(v_mother,num);
          SET  num = num + 1;
      END IF;
  END
```

下面是一个基于 ORACLE 数据库的缴费存储过程的例子。

例 5-12　ORACLE 存储过程应用示例。

```
CREATE OR REPLACE PROCEDURE p_update_id  --更新计费记录存储过程
      (v_tele_num in cash_charge_table.tele_num%type,
      v_id_no in cash_charge_table.id_no%type,
      v_pay_status in cash_charge_table.pay_status%type,
      v_process_no in cash_charge_table.process_no%type,
      v_operator_no in cash_charge_table.operator_no%type,
      v_spot_pay_method in cash_charge_table.spot_pay_method%type,
      v_nomal_charge_fare in operator_table.nomal_charge_fare%type,
      v_total_owing_fare in operator_table.total_owing_fare%type,
      v_total_fine_fare in operator_table.total_fine_fare%type,
      v_coin in operator_table.coin%type)
IS
```

```
    total_owing_fare cash_charge_table.total_fare%type;
BEGIN
  if v_total_owing_fare > 0 then--停复机处理
       if v_pay_status = 'S' then
        select nvl(sum(total_fare),0) into total_owing_fare from cash_
charge_table
        where tele_num = v_tele_num and fee_type = 'C' and nvl(owing_flag,
'') = 'Y' and nvl(pay_flag,' ') <> 'Y' and pay_method_rollback = 'X';
        if total_owing_fare = 0 then
      stop_recover_process(v_id_no,v_tele_num,'0',SYSDATE,v_operator_no);
             --0 为复机。调用存储过程 stop_recover_process
        end if;
       end if;
      if v_pay_status = 'C' then
        if total_owing_fare = 0 then
      stop_recover_process(v_id_no,v_tele_num,'0',SYSDATE,v_operator_no);
        end if;
      end if;
      end if;

UPDATE cash_charge_table--更新欠费——滞纳金两个月起计
SET fine_fare = decode(sign(to_char(add_months(account_date,2),'yyyymmdd')
-'20070401'),-1,trunc(total_fare*free_degree*trunc(to_date('2007/04/01',
'yyyy/mm/dd')-add_months(account_date,2),0)/100,1)+trunc(total_fare*
        free_degree*trunc(sysdate-to_date('2007/04/01','yyyy/mm/dd'),0)*3/
1000,1),
          1,decode(sign(sysdate-dd_months(account_date,2)),-1,0,1,1)*trunk
      (total_fare*free_degree*trunc(sysdate-add_months(account_date,2),0)
         *3/1000,1)), process_no = v_process_no, operator_no = v_operator_no,
          pay_date = SYSDATE, spot_pay_method = v_spot_pay_method,
          pay_status = v_pay_status, pay_flag = 'Y'
     WHERE tele_num = v_tele_num  AND id_no = v_id_no
          AND (trunc(months_between(SYSDATE,account_date),0) >= 1 or
                  nvl(owing_flag,' ')='Y')  AND fee_type = v_pay_status
          AND nvl(pay_flag,' ') <> 'Y' AND pay_method_rollback = 'X';

     UPDATE cash_charge_table    --更新当月费
     SET process_no = v_process_no,  operator_no = v_operator_no,
          pay_date = sysdate,  spot_pay_method = v_spot_pay_method,
          pay_status = v_pay_status, pay_flag = 'Y'
      WHERE tele_num = v_tele_num  AND id_no = v_id_no
         AND trunc(months_between(SYSDATE,account_date),0) = 0
         AND fee_type = v_pay_status  AND nvl(owing_flag, ' ') = ' '
         AND nvl(pay_flag,' ') = ' '  AND pay_method_rollback = 'X';
      INSERT INTO operator_table(operator_no,   --在营业员个人表中插入一条记录
  tele_num,process_no,pay_date,pay_status,spot_pay_method,
  nomal_charge_fare, total_owing_fare,total_fine_fare,coin,pay_status_2)
       VALUES(v_operator_no,v_tele_num,v_process_no,sysdate, v_pay_status,
            v_spot_pay_method,v_nomal_charge_fare, v_total_owing_fare,
```

```
           v_total_fine_fare,v_coin,v_pay_status);
EXCEPTION
       WHEN others THEN
          raise_application_error(-20109,sqlerrm);   --报错提示
END p_update_id;
```

5.3　存储过程、函数、触发器和复制：常见问题

（1）MySQL 5.1 存储过程和函数对复制起作用吗？

起作用。在存储过程和函数中被执行标准行为可以被从主 MySQL 服务器复制到从服务器，但有少数限制。MySQL 复制基于主服务器在二进制日志中跟踪所有对数据库的更改（更新、删除等）。因此，要进行复制，必须在主服务器上启用二进制日志。

每个从服务器从主服务器接收主服务器已经记录到其二进制日志保存的更新，以便从服务器可以对其数据复制执行相同的更新。

（2）在主服务器上创建的存储过程和函数可以被复制到从服务器上吗？

可以。通过一般 DDL 语句执行的存储过程和函数，在主服务器上的创建被复制到从服务器，其目标将存在两个服务器上。同时对存储过程和函数的 ALTER 和 DROP 语句也被复制。

（3）行为如何在已复制的存储过程和函数中发生？

MySQL 记录每个发生在存储过程和函数中的 DML 事件，并复制这些单独的行为到从服务器。

（4）对一起使用存储过程、函数和复制有没有什么特别的安全要求？

有要求。因为一个从服务器有权限来执行任何读自主服务器的二进制日志的语句，指定的安全约束因与复制一起使用的存储过程和函数而存在。如果复制或二进制日志大体上是激活的（为 point-in-time 恢复的目的），那么 MySQL DBA 有两个安全选项可选：

- 任何想创建存储过程的用户必须被赋予 SUPER 权限。
- 作为选择，一个 DBA 可以设置 log_bin_trust_routine_creators 系统变量为 1，它将会允许有 CREATE ROUTINE 权限的用户来创建一个存储过程和函数。

（5）触发器对复制起作用吗？

MySQL 5.1 中的触发器和复制就像在大多数其他数据库引擎中一样工作，在这些引擎中，通过触发器在主服务器上执行的行为不被复制到从服务器。取而代之的是，位于主 MySQL 服务器表中的触发器需要在那些存在于任何 MySQL 从服务器上的表内被创建，以便于触发器也可以在从服务器上被激活。

（6）一个行为如何通过从主服务器上复制到从服务器上的触发器来执行呢？

首先，主服务器上的触发器必须在从服务器上重建。一旦重建了，复制流程就像其他参与到复制中的标准 DML 语句一样工作。例如，考虑一个已经插入触发器 AFTER 的 DEPT 表，它位于主 MySQL 服务器上。同样地，DEPT 表和 AFTER 插入触发程序也存在于从服务器上。复制流程可能是这样的：

① 对 DEPT 做一个 INSERT 语句操作。

② DEPT 上的 AFTER 触发器被激活。

③ INSERT 语句被写进二进制日志。

④ 从服务器上的复制拾起 INSERT 语句给 DEPT 表，并在从服务器上执行它。

⑤ 位于从服务器 DEPT 上的 AFTER 触发器被激活。

（7）如果主服务器正在运行并且不想停止主服务器，怎样配置一个从服务器？

可以有多种方法。如果在某时间点做过主服务器备份并且记录了相应快照的二进制日志名和偏移量（通过 SHOW MASTER STATUS 命令输出），采用下面的步骤：

① 确保从服务器分配了一个唯一的服务器 ID 号。

② 在从服务器上执行下面的语句，为每个选项填入适当的值：

```
CHANGE MASTER TO
  MASTER_HOST='master_host_name',
  MASTER_USER='master_user_name',
  MASTER_PASSWORD='master_pass',
  MASTER_LOG_FILE='recorded_log_file_name',
  MASTER_LOG_POS=recorded_log_position;
```

③ 在从服务器上执行 START SLAVE 语句。

5.4 触 发 器

触发器是一种特殊的存储过程，它在插入、删除或修改特定表中的数据时触发执行，它比数据库本身标准的功能有更精细和更复杂的数据控制能力。与存储过程不同的是，存储过程通过其他程序来启动运行，而触发器是由一个事件来启动运行。并且触发器不能接收参数。数据库触发器有以下作用：

- 安全性。可以基于数据库的值使用户具有操作数据库的某种权利。可以基于时间限制用户的操作，例如，不允许下班后和节假日修改数据库数据。可以基于数据库中的数据限制用户的操作，例如，不允许某商品的价格一次下降超过 20%。
- 审计。可以跟踪用户对数据库的操作。审计用户操作数据库的语句。把用户对数据库的更新写入审计表。
- 实现复杂的数据完整性规则。实现非标准的数据完整性检查和约束。触发器可产生比规则更为复杂的限制。与规则不同，触发器可以引用列或数据库对象。例如，触发器可以回退任何卖出去的商品超过库存的数量。
- 实现复杂的非标准的数据库相关完整性规则。触发器可以对数据库中相关的表进行连环更新。例如，在 student 表 code 列上删除触发器，可导致同步删除在其他表中与之匹配的数据。
- 同步实时地复制表中的数据。
- 自动计算数据值。如果数据的值达到了一定的要求，则进行特定的处理。例如，财务某个指标低于 100 万元，则立刻向公司高层发出警告数据。

5.4.1　创建触发器

只有具备 Super 权限的 MySQL 用户才能执行创建触发器的命令。创建触发器的命令格式如下：

```
CREATE TRIGGER trigger_name BEFORE|AFTER INSERT|UPDATE|DELETE
ON table_name FOR EACH ROW code
```

触发器是与表有关的命名数据库对象，当表上出现特定事件时，将激活该对象。

触发器与命名为 table_name 的表相关。table_name 必须引用永久性表。MySQL 中不能将触发器与临时表或视图关联起来。

触发器的触发事件可以是下列之一。

- INSERT：将新行插入表时激活触发程序，例如，通过 INSERT、LOAD DATA 和 REPLACE 语句。

创建数据库表 player：

```
CREATE TABLE player
   (user              VARCHAR(20) NOT NULL,
   time1              TIMESTAMP NOT NULL,
   player_no          SMALLINT NOT NULL,
   type               VARCHAR(1) NOT NULL,
   player_no_new      INT,
   primary key         (user,player_no) )
```

创建 INSERT 触发器如例 5-13 所示。

例 5-13　创建 INSERT 触发器。

```
CREATE TRIGGER player_insert
   AFTER INSERT ON player FOR EACH ROW
   BEGIN
      INSERT INTO player (user,time1,player_no,type,player_no_new)
      VALUES (user,CURDATE(),NEW.player_no,'钢琴',NULL) ;
   END
```

- UPDATE：更改某一行时激活触发器，例如，通过 UPDATE 语句。

例 5-14　创建 UPDATE 触发器。

```
CREATE TRIGGER player_update
   BEFORE UPDATE ON player FOR EACH ROW
   BEGIN
      IF NEW. player_no_new < 0 THEN
         SET NEW.player_no_new = 0;
      ELSEIF NEW. player_no_new > 1000 THEN
         SET NEW.player_no_new = 1000;
      END IF;
   END;
```

- ❑ DELETE：从表中删除某一行时激活触发器，例如，通过 DELETE 和 REPLACE 语句。

例 5-15　创建 DELETE 触发器。

```
CREATE TRIGGER player_delete
   AFTER DELETE ON player FOR EACH ROW
   BEGIN
      DELETE FROM player
      WHERE player_no = OLD.player_no;
   END
```

【特别提示】触发事件与以表操作方式激活触发器的 SQL 语句并不很类似，这点很重要。例如，关于 INSERT 的 BEFORE 触发器不仅能被 INSERT 语句激活，也能被 LOAD DATA 语句激活。

可能会造成混淆的例子之一是 INSERT INTO…ON DUPLICATE UPDATE…语法：BEFORE INSERT 触发器对于每一行将激活，后跟 AFTER INSERT 触发器，或 BEFORE UPDATE 和 AFTER UPDATE 触发器，具体情况取决于行上是否有重复键。

对于具有相同触发器动作时间和事件的给定表，不能有两个触发器。例如，对于某一表，不能有两个 BEFORE UPDATE 触发器，但可以有一个 BEFORE UPDATE 触发器和一个 BEFORE INSERT 触发器，或一个 BEFORE UPDATE 触发器和一个 AFTER UPDATE 触发器。

code 是当触发程序激活时执行的语句。如果打算执行多个语句，可使用 BEGIN…END 复合语句结构。这样就能使用存储子程序中允许的相同语句。

另外，触发器中也能调用存储过程。首先创建一个存储过程：

```
CREATE PROCEDURE player_insert
    (IN  pno        INT,
     IN  ptype      VARCHAR(1),
     IN  pno_new    INT)
BEGIN
   INSERT INTO player(user,time1,player_no,type,player_no_new)
   VALUES(user,CURDATE(),pno,ptype,pno_new);
END
```

接着创建一个触发器：

```
CREATE TRIGGER player_insert
   AFTER INSERT ON player FOR EACH ROW
   BEGIN
      CALL player_insert(NEW.player_no,'小提琴',NULL);--调用存储过程
   END
```

5.4.2　删除触发器

删除触发器的命令格式如下：

```
DROP TRIGGER [database_name.]trigger_name
```

数据库名称是可选的。如果省略了 database_name，将从当前方案中删除触发器。

【特别提示】从 MySQL 5.0.10 之前的 MySQL 版本升级到 5.0.10 或更高版本时（包括所有的 MySQL 5.1 版本），必须在升级之前删除所有的触发器，并在随后重新创建它们；否则，在升级之后 DROP TRIGGER 不能工作。另外，使用 DROP TRIGGER 语句需要 SUPER 权限。

5.4.3　触发器应用示例

下面列出 MySQL 数据库触发器和 ORACLE 触发器的两个例子，比较两个数据库触发器的异同点。

例 5-16　MySQL 触发器示例。

```
CREATE TRIGGER max1
 AFTER INSERT,UPDATE(POSITION) OF COMMITTEE_MEMBERS
       FOR EACH ROW
   BEGIN
     SELECT COUNT(*)
     INTO number_members
     WHERE playerno IN
           (SELECT  playerno
            FROM   committee_members
            WHERE  current date BETWEEN BEGIN_DATE AND END_DATE
            GROUP BY POSITION
            HAVING COUNT(*)>1)
    IF number_members > 0 THEN
       ROLLBACK WORK;
    ENDIF
 END
```

例 5-17　ORACLE 触发器示例。

```
    CREATE TRIGGER TRIG_TRI_TROUBLE_STATUS
       AFTER INSERT ON TR_TROUBLE FOR EACH ROW  --对每行插入之后触发
    DECLARE
       CABLE_NAME VARCHAR2(19);
       ERR_REASON VARCHAR2(40);
       ERR_REF VARCHAR2(1);
       CHANNEL_A VARCHAR2(10);
       CHANNEL_B VARCHAR2(10);
       CHANNEL_C VARCHAR2(10);
       ERR_TYPE VARCHAR2(1);
    BEGIN
      SELECT ERR_REASON,ERR_REF INTO ERR_REASON,ERR_REF
      FROM TR_TROUBLE
      WHERE CIRCUIT_NAME=:NEW.CIRCUIT_NAME;
```

```
    SELECT ERR_TYPE INTO ERR_TYPE
    FROM TR_TROUBLE_REASON
    WHERE ERR_REASON=ERR_REASON;
    IF ERR_REASON='A' OR ERR_REASON='B' OR ERR_REASON='C' THEN
       --将相应光缆的状态改变
      SELECT DISTINCT CABLE_NAME INTO CABLE_NAME
      FROM TR_BEAM
             WHERE BEAM_ID IN(SELECT BEAM_ID
                                 FROM TR_DL_BEAM
                                 WHERE CIRCUIT_ID IN(SELECT CIRCUIT_ID
                                 FROM TR_DL_CIRCUIT
                                 WHERE CIRCUIT_NAME=
                                      :NEW.CIRCUIT_NAME));
       IF CABLE_NAME IS NOT NULL THEN
         IF ERR_REF='1' THEN
           UPDATE TR_CABLE
           SET CABLE_STATUS='E'
           WHERE CABLE_NAME=CABLE_NAME;
        ELSE
           UPDATE TR_CABLE
           SET CABLE_STATUS='D'
           WHERE CABLE_NAME=CABLE_NAME;
         END IF;
       END IF;
    ELSE
     --将相应设备的状态改变
        SELECT CHANNEL_A,CHANNEL_B,CHANNEL_C
        INTO CHANNEL_A,CHANNEL_B,CHANNEL_C
        FROM TR_TROUBLE
        WHERE CIRCUIT_NAME=:NEW.CIRCUIT_NAME;

        TR_LET_CHANNEL_DEVICE(CHANNEL_A,'1');
        TR_LET_CHANNEL_DEVICE(CHANNEL_B,'1');
        TR_LET_CHANNEL_DEVICE(CHANNEL_C,'1');
    END IF;
  END
```

5.5　存储过程和触发器的二进制日志功能

二进制日志以一种更有效的格式，并且是事务安全的方式包含更新日志中可用的所有信息。

二进制日志包含了所有更新了的数据或者已经潜在更新了的数据（例如，没有匹配任何行的一个 DELETE）的所有语句。该语句以“事件”的形式保存，它描述数据更改情况。

【特别提示】二进制日志已经代替了老的更新日志，更新日志在 MySQL 5.1 中不再使用。

二进制日志有两个重要目的：

- 要进行复制，必须在主服务器上启用二进制日志

复制的基础是主服务器发送包含在二进制日志中的事件到从服务器，从服务器执行这些事件来造成与对主服务器造成的同样的数据改变。MySQL 复制基于主服务器在二进制日志中跟踪所有对数据库的更改（更新、删除等）。

- 特定的数据恢复操作必须要使用二进制日志

备份的文件被恢复之后，备份后记录的二进制日志中的事件被重新执行。这些事件把数据库从备份点的时间带到当前。例如，现在假设在星期二上午 10 点出现了数据库灾难性崩溃，需要使用备份文件进行恢复。恢复时，首先恢复最后的完全备份（从星期日下午 5 点开始）。完全备份文件是一系列 SQL 语句，因此恢复它很容易：

```
shell> mysql < backup_sunday_1_pm.sql
```

在该点，数据恢复到星期日下午 5 点的状态。要想恢复从那时起的更改，必须使用增量备份，也就是 gbichot2-bin.000001 和 gbichot2-bin.000002 二进制日志文件。根据需要从备份处取过这些文件，然后按下述方式处理：

```
shell> mysqlbinlog gbichot2-bin.000001 gbichot2-bin.000002 | mysql
```

现在将数据恢复到星期一下午 2 点的状态，但是从该时刻到崩溃之间的数据仍然有丢失。要想恢复，需要 MySQL 服务器将 MySQL 二进制日志保存到安全的位置，应与数据文件的保存位置不同的地方，保证这些日志不在毁坏的硬盘上（也就是说，可以用--log-bin 选项启动服务器，指定一个其他物理设备上的与数据目录不同的位置。这样，即使包含该目录的设备丢失，日志也不会丢失）。如果执行了这些操作，就会有 gbichot2-bin.000003 文件，可以用它来恢复大部分最新的数据更改，而不会丢失到崩溃时的数据。

如果二进制日志功能不被允许，复制将不可能，为数据恢复的二进制日志也不存在。

在 MySQL 中，以存储过程和触发器的二进制日志功能也引发了很多问题，这里不再描述，读者可参考最新的 MySQL 参考手册。

5.6　小　　结

使用存储过程可以在数据库中进行程序的开发。应用程序能够根据所需调用这些代码；同时，使用存储过程可以减少网络流量。

存储过程可以返回一个或多个 SELECT 结果，它可以通过 CREATE PROCEDURE 语句进行创建；而存储函数只能返回一个值，使用 CREATE FUNCTION 语句可以创建函数。

触发器是一种特殊的存储过程，与存储过程不同的是，存储过程通过其他程序来启动运行，而触发器是由一个事件来启动运行，它在表的数据变化时发生作用。触发器可以维护数据的完整性。MySQL 5.1 支持 3 种类型触发器：INSERT、UPDATE 和 DELETE。当向表中插入数据、更新数据或删除数据时，触发器就被调用。

简单地介绍了存储过程和触发器的二进制日志功能。二进制日志以一种更有效的格式，包含更新日志中可用的所有信息。进行数据备份和数据恢复都要通过二进制日志来实现。

第 6 章　JDBC 基础

本章从 JDBC 的一些基本知识入手，引领读者进入基于 MySQL 数据库 Java 编程的世界。本章介绍 JDBC 的基本概念，包括 JDBC 基本功能、JDBC 驱动的 4 种基本类型等；同时对传统的 ODBC 接口的体系结构以及数据源的配置方法进行介绍，阐述了 JDBC 与 ODBC 的异同，旨在加深读者对 JDBC 的了解。在本章的最后，对 JDBC API 进行详尽的介绍，当然其具体的应用将在以后的章节向读者逐步展开。

读者在学习本章时要从层次性和安全性等多个角度来理解 JDBC 数据库设计的方法，以便以后章节的学习能驾轻就熟，在实际工程中也能更好地运用。

6.1　基本的 JDBC 概念

JDBC 是 Java Database Connectivity（Java 数据库连接）的英文缩写，顾名思义，在 Java 程序中，可以利用它来建立数据库连接，执行 SQL 语句，并对数据进行处理。从本质上来说它是一个基于 Java 的面向对象的应用编程接口，描述了一套访问关系数据库的标准 Java 类库。

6.1.1　JDBC 基本功能

JDBC 的主要功能有：

（1）与数据库建立连接。

（2）发送 SQL 语句。

（3）处理数据库返回结果。

6.1.2　JDBC 的层次

JDBC 首先为数据库应用开发人员、数据库前台工具开发人员提供了一种标准的 API，使开发人员可以用纯 Java 语言编写完整的数据库应用程序；其次，它还为数据库厂商提供了一个标准体系结构，让厂商可以为自己的数据库产品提供 JDBC 驱动程序，从而提高了 Java 程序访问数据库的效率。从中可以看出 JDBC 接口的两个层次：一个是面向程序开发人员的 JDBC API，另一个是底层的 JDBC Driver API。

JDBC API 被描述成为抽象的 Java 接口，它的应用程序可以打开某个数据库连接，执行 SQL 语句并且处理结果，将会在下面的章节中对这方面进行详细的介绍。

6.1.3　JDBC 驱动

JDBC 的总体结构由 4 个组件构成，它们是：应用程序、驱动程序管理器、驱动程序和数据源，如图 6-1 所示。

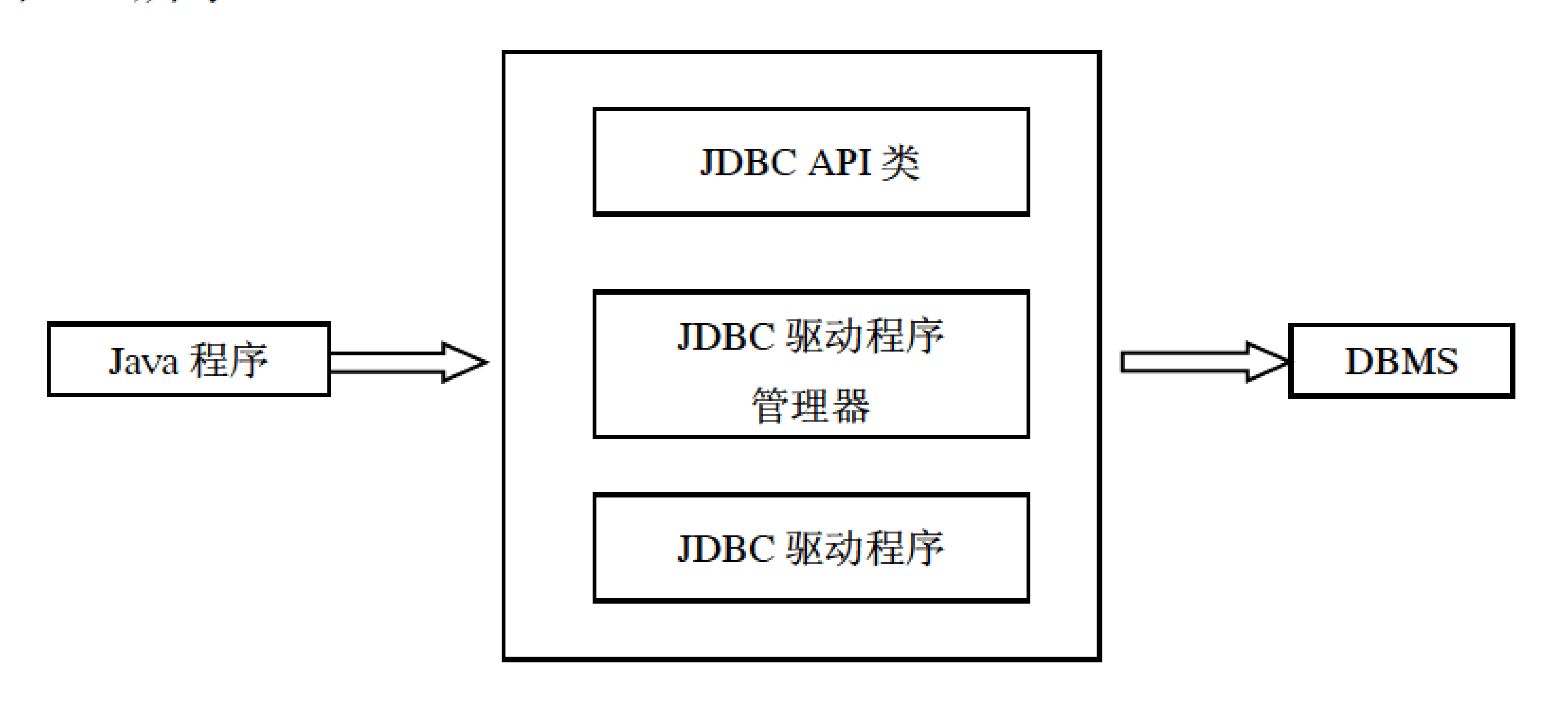


图 6-1　JDBC 总体结构图

（1）应用程序：用于发送或者接收数据。

（2）驱动程序管理器：处理数据源相应的驱动程序。

（3）驱动程序：提供数据源和应用程序之间的接口。

（4）数据源：SQL 兼容数据库。

应用程序只需调用 JDBC API，而由 JDBC 实现层（即 JDBC 驱动程序）去处理与数据库的通信，从而让应用程序不再受限于具体的数据库产品。如果想通过 JDBC 去连接某个特定的数据库系统，只需使用专门为这种数据库系统而开发的 JDBC 驱动程序，那么这些相关的 JDBC 驱动应该从哪里获得呢？实际上各大数据库厂商（Oracle、Sybase、DB2 等）对 JDBC 都有很好的支持，它们的官方网站都提供了各种版本的下载。在第 7 章中，将学习 Connector/J 的相关知识，Connector/J 就是 MySQL 数据库的 JDBC 驱动，学完之后，将会对这一概念有更具体的体会。

JavaSoft 将 JDBC 驱动程序细分为 4 大类，分别是 JDBC/ODBC Bridge、Native API Partly Java、Net Protocol All Java 和 Native Protocol All Java。

Type1：JDBC-ODBC 桥

如图 6-2 所示，JDBC-ODBC 桥由 SUN 公司提供，是 JDK 提供的标准 API。因为微软推出的 ODBC 比 JDBC 出现的时间早，且应用广泛，支持绝大多数的数据库，当 SUN 公司推出 JDBC 时，为了支持更多的数据库，Intersolv 和 Java Soft 联合开发 JDBC-ODBC 桥。这种类型的驱动实际是把所有 JDBC 的调用传递给 ODBC，再由 ODBC 调用本地数据库驱动代码。只要本地机装有相关的 ODBC 驱动，那么采用 JDBC-ODBC 桥几乎可以访问所有的数据库，JDBC-ODBC 方法对于客户端已经具备 ODBC Driver 的应用还是可行的。但由于 JDBC-ODBC 先调用 ODBC，再由 ODBC 去调用本地数据库接口访问数据库，需要经过多层调用，所以执行效率比较低，对于那些大数据量存取的应用是不适合的。而且这种方法要求客户端必须安装 ODBC 驱动，所以对于基于 Internet、Intranet 的应用是不现实的。

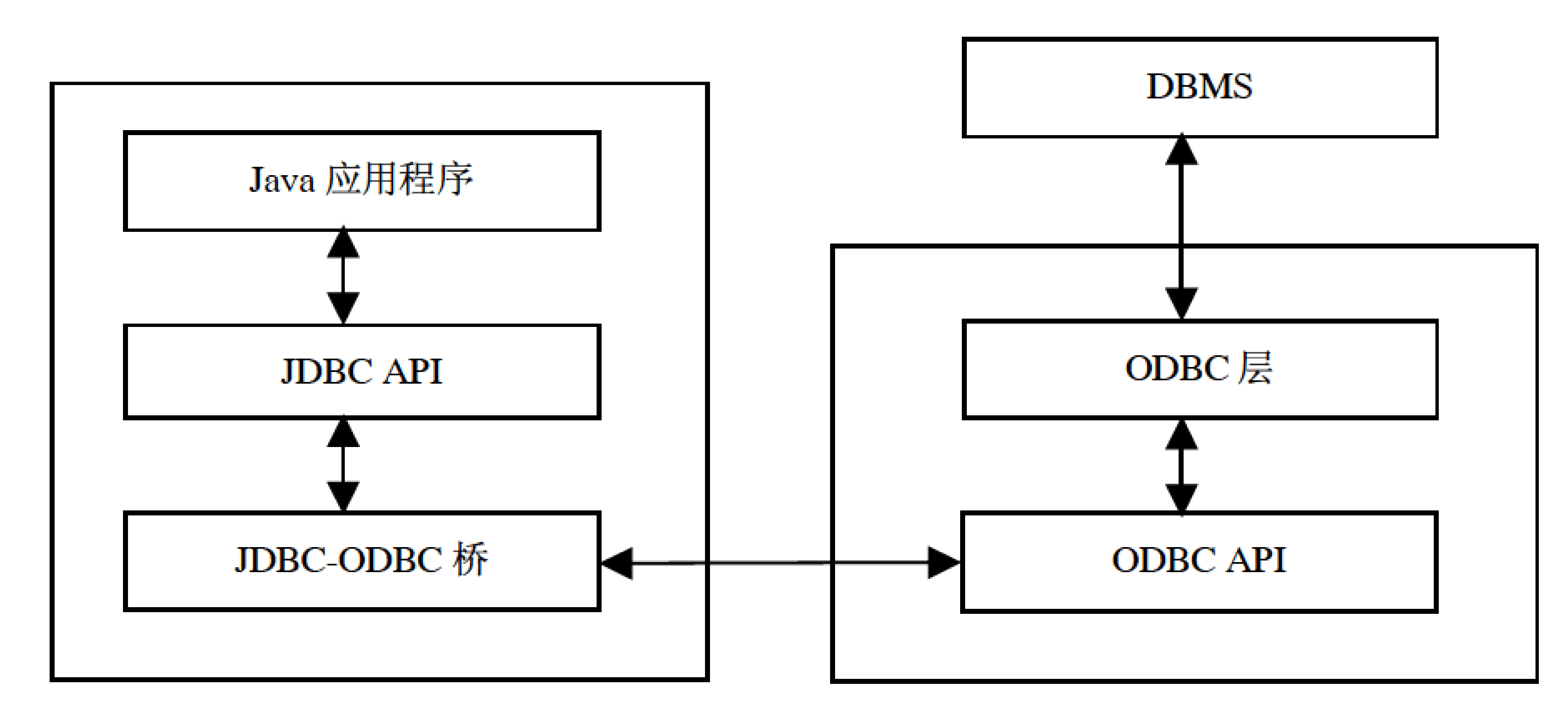


图 6-2　JDBC-ODBC 桥

JDBC-ODBC 桥比“纯”ODBC 多了几项优势：ODBC API 主要面向 C/C++ 程序员，它使 Java 程序员无须应付非 Java 概念。ODBC API 非常复杂，它把高级功能和低级函数混合起来。而 JDBC API 相对来说简单易学，因此 JDBC-ODBC 桥使程序员可以依赖 JDBC API。JDBC-ODBC 桥允许程序通过标准化 JDBC 接口处理 ODBC。在提出更好的解决方案时，这就可以使程序避免束缚到 ODBC 上。

尽管 JDBC-ODBC 桥应被看成是过渡性解决方案，不过，在数据库没有提供 JDBC 驱动，只有 ODBC 驱动的情况下，也只能采用 JDBC-ODBC 桥的方式访问数据库。例如，对微软的 Access 数据库操作时，就只能用 JDBC-ODBC 桥来访问了。

Type2：本地 API Java 驱动程序

大部分数据库厂商提供与他们的数据库产品进行通信所需要的 API，这些 API 往往用 C 语言编写，依赖于具体的平台，本地 API Java 驱动程序通过 JDBC 驱动程序将应用程序中的调用请求转化为本地 API 调用，由本地 API 与数据库通信，数据库处理完请求将结果通过本地 API 返回，进而返回给 JDBC 驱动程序，JDBC 驱动程序将返回的结果转化为 JDBC 标准形式，再返回给客户程序，其工作原理如图 6-3 所示。

这种 JDBC 驱动的优点是减少了 ODBC 的调用环节，提高了数据访问的效率，并且能够充分利用厂商提供的本地 API 功能，但前提是需要在客户的机器上安装本地 JDBC 驱动程序和特定数据库厂商的本地 API，这样就不适合基于 Internet 的应用，并且它的执行效率比起 Type3 和 Type4 型的 JDBC 驱动还是不够高。

Type3：网络纯 Java 驱动程序

这种驱动程序将 JDBC 转换为与 DBMS 无关的网络协议，之后这种协议又被某个服务器转换为一种 DBMS 协议，如图 6-4 所示。

这种网络服务器中间件能够将它的纯 Java 客户机连接到多种不同的数据库上。所用的具体协议取决于提供者。通常，这是最为灵活的 JDBC 驱动程序，能提供适合于 Intranet 用的产品。为了使这些产品也支持 Internet 访问，它们必须处理 Web 所提出的安全性、通过防火墙的访问等方面的额外要求。

这种驱动实际上是根据我们熟悉的三层结构建立的。JDBC 先把对数局库的访问请求传递给网络上的中间件服务器。中间件服务器再把请求翻译为符合数据库规范的调用，再把

这种调用传给数据库服务器。Bea 公司的 WebLogic 和 IBM 公司的 Websphere 应用服务器就包含了这种类型的驱动。由于这种驱动是基于 Server 的，所以，它不需要在客户端加载数据库厂商提供的代码库，而且它在执行效率和可升级性方面是比较好的。因为大部分功能实现都在 Server 端，所以这种驱动可以设计得很小，可以非常快速地加载到内存中。但是，这种驱动在中间件层仍然需要配置其他数据库驱动程序，并且由于多了一个中间层传递数据，从某种意义上说，它的执行效率不是最好的。

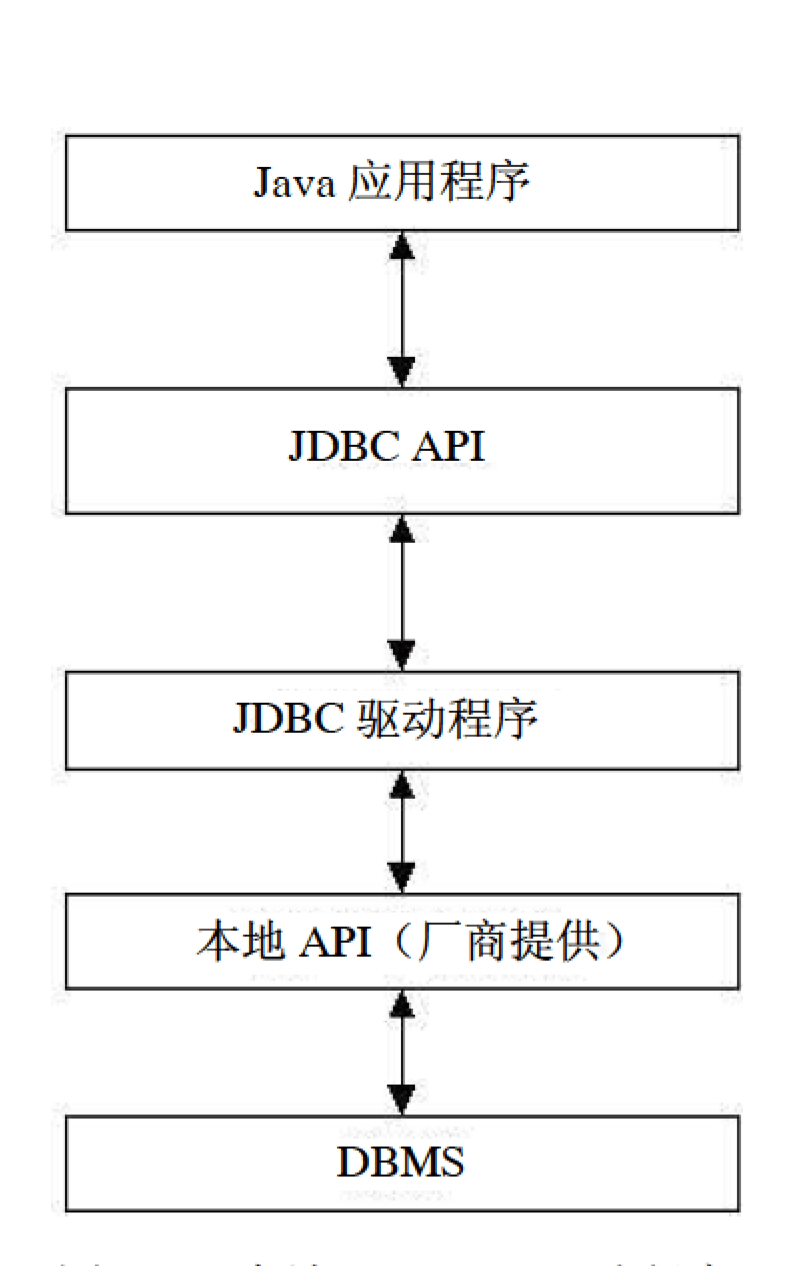


图 6-3　本地 API Java 驱动程序

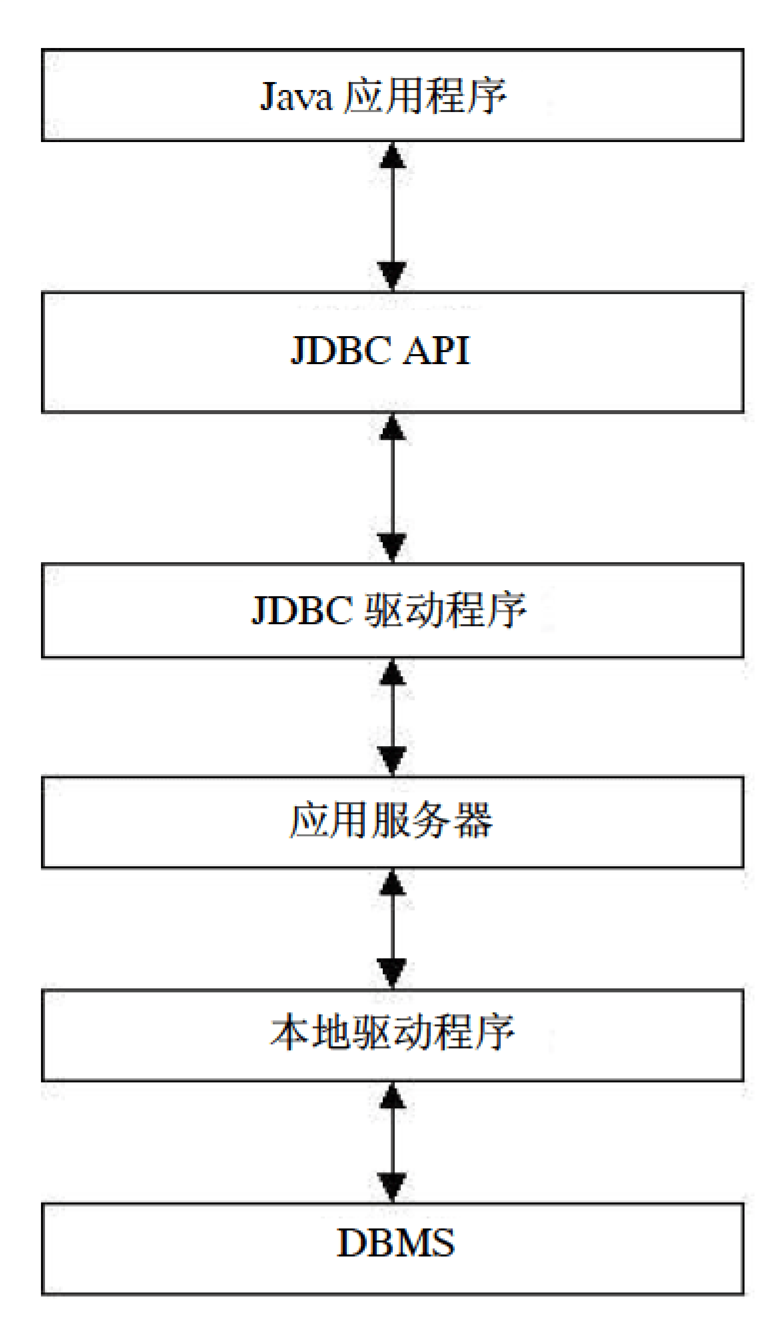


图 6-4　JDBC 网络纯 Java 驱动程序

Type4：本地协议纯 Java 驱动程序

这种类型的驱动程序将 JDBC 调用直接转换为 DBMS 所使用的网络协议。这种驱动与数据库建立直接的套接字连接，采用具体数据库厂商的网络协议把 JDBC API 调用转换为直接网络调用，也就是允许从客户机机器上直接调用 DBMS 服务器，是 Intranet 访问的一个很实用的解决方法。由于 Type4 驱动写的应用可以直接和数据库服务器通信。这种类型的驱动完全由 Java 实现，因此实现了平台独立性，如图 6-5 所示。

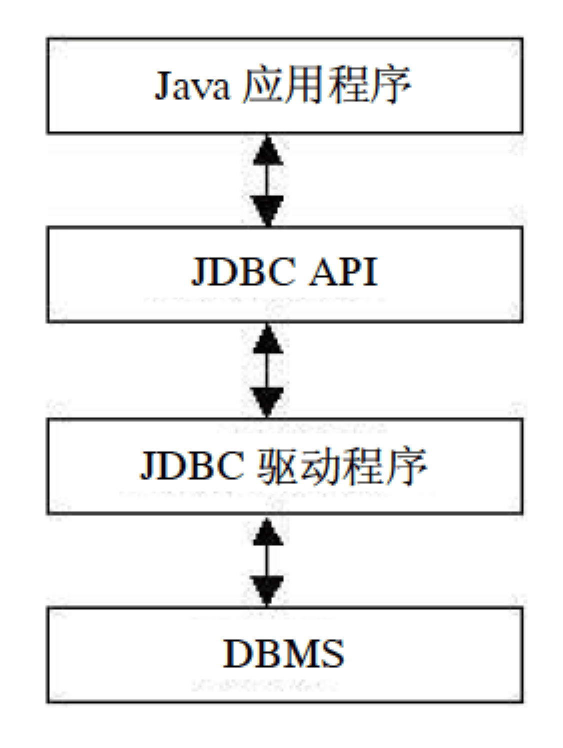


图 6-5　本地协议纯 Java 驱动程序

建议尽可能地使用纯 Java JDBC 驱动程序代替桥和 ODBC 驱动程序，这可以完全省去 ODBC 所需的客户机配置，也免除了 Java 虚拟机被桥引入的本地代码（即桥本地库、ODBC 驱动程序管理器库、ODBC 驱动程序库和数据库客户机库）中的错误所破坏的可能性。

6.2　关于 ODBC

关系型数据库产生后很快就成为数据库系统的主流产品，由于每个 DBMS 厂商都有自己的一套标准，早期对数据库的访问都是调用数据库厂商提供的专有 API，例如，为了访问 Sybase 数据库需要专门写一个程序，为访问 Oracle 数据库又专门另写一个程序。人们很早就产生了标准化的想法，于是产生了 SQL，由于其语法规范逐渐被人们所接受，成为 RDBMS 上的主导语言。最初，各数据库厂商为了解决互连的问题，往往提供嵌入式 SQL API，用户在客户机端要操作系统中的 RDBMS 时，往往要在程序中嵌入 SQL 语句进行预编译。由于不同厂商在数据格式、数据操作、具体实现甚至语法方面都具有不同程度的差异，所以彼此不能兼容。长期以来，这种 API 的非规范情况令用户和 RDBMS 厂商都不能满意。一个特定的前端应用却不能访问不同数据库上的数据，主要原因有两个：

一是各厂商的 SQL 版本不同，每个关系数据库管理系统（RDBMS）厂商都对标准 SQL 进行了独特的扩充或解释，使得不同的 RDBS 提供的 SQL 互不兼容；二是不同厂商的 RDBMS 在客户机与数据库服务器之间使用了不同的通信协议。

6.2.1　ODBC 接口

正是在这种技术背景下，为了在 Windows 平台下提供统一的数据库访问方式，Microsoft 公司于 1992 年推出了 ODBC 产品，并提供 ODBC API，使应用程序与 DBMS 在逻辑上分离，应用程序具体数据库无关性使用者在程序中只需要调用 ODBC API，由 ODBC 驱动程序将调用请求转换为对特定数据库的调用请求，这样同一个应用程序就可以访问不同的数据库系统，存取多个数据库中的数据，提高了应用程序的可移植性。与嵌入式 SQL 相比，ODBC 一个最显著的优点是用它生成的应用程序与数据库或数据库引擎无关。ODBC 工作原理示意图如图 6-6 所示。

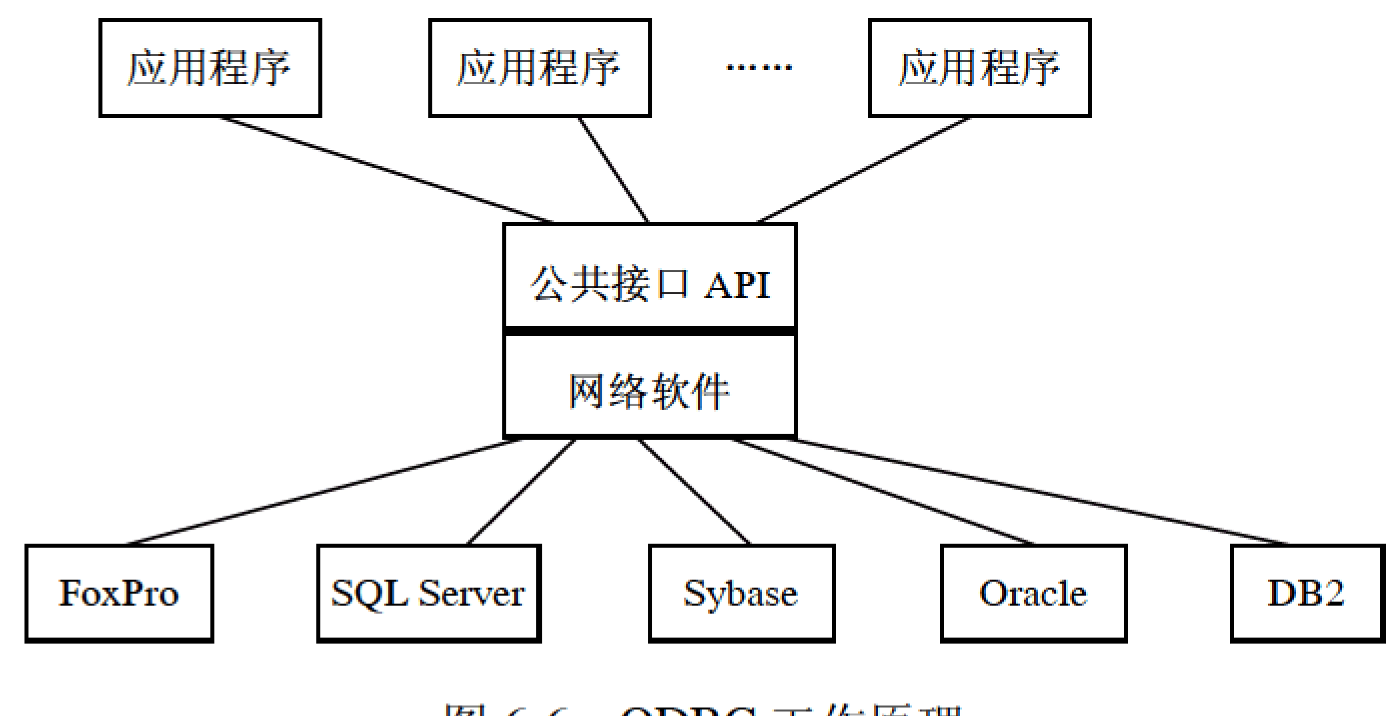


图 6-6　ODBC 工作原理

在传统方式中，开发人员要熟悉多个 DBMS 及其 API，一旦 DBMS 端出现变动，则往往导致用户端系统重新编建或者源代码的修改，这给开发和维护工作带来了很大困难。在 ODBC 方式中，不管底层网络环境如何，也无论采用何种 DBMS，用户在程序中都使用同一套标准代码，无须逐个了解各 DBMS 及其 API 的特点，源程序不因底层的变化而重新编译或修改，从而减轻了开发维护的工作量，缩短了开发周期。

由于 ODBC 思想上的先进性，受到了众多厂家与用户的青睐，成为一种广为接受的标准。目前，已经有 130 多家独立厂商宣布了对 ODBC 的支持，常见的 DBMS 都提供了 ODBC 的驱动接口，这些厂商包括 Oracle、Sybase、Informix、Ingres、IBM（DB/2）、DEC（RDB）、HP（ALLBASE/SQL）、Borland（Paradox）等。目前，ODBC 是客户机/服务器系统中的一个重要支持技术。

概括起来，ODBC 具有以下灵活的特点：

（1）使用户程序有很高的互操作性，相同的目标代码适用于不同的 DBMS。

（2）由于 ODBC 的开放性，它为程序集成提供了便利，为客户机/服务器结构提供了技术支持。

（3）由于应用与底层网络环境和 DBMS 分开，简化了开发维护上的困难。

6.2.2　ODBC 体系结构

ODBC 技术为应用程序提供了一套 CLI（Call-Leve Interface，调用层接口）函数库和基于 DLL（Dynamic Link Library，动态链接库）的运行支持环境。使用 ODBC 开发数据库应用程序时，在应用程序中调用标准的 ODBC 函数和 SQL 语句，通过可加载的驱动程序将逻辑结构映射到具体的 DBMS 或者应用系统所使用的系统。换言之，连接其他数据库和存取这些数据库的低层操作由驱动程序驱动各个数据库完成。

ODBC 的卓越贡献是使应用程序具有良好的互用性和可移植性，并且具备同时访问多种 DBMS 的能力，从而克服了传统数据库应用程序的缺陷。对用户来说，ODBC 驱动程序屏蔽掉了不同的 DBMS 的差异。

ODBC 是一个分层的体系结构，这样可保证其标准性和开放性，如图 6-7 所示。

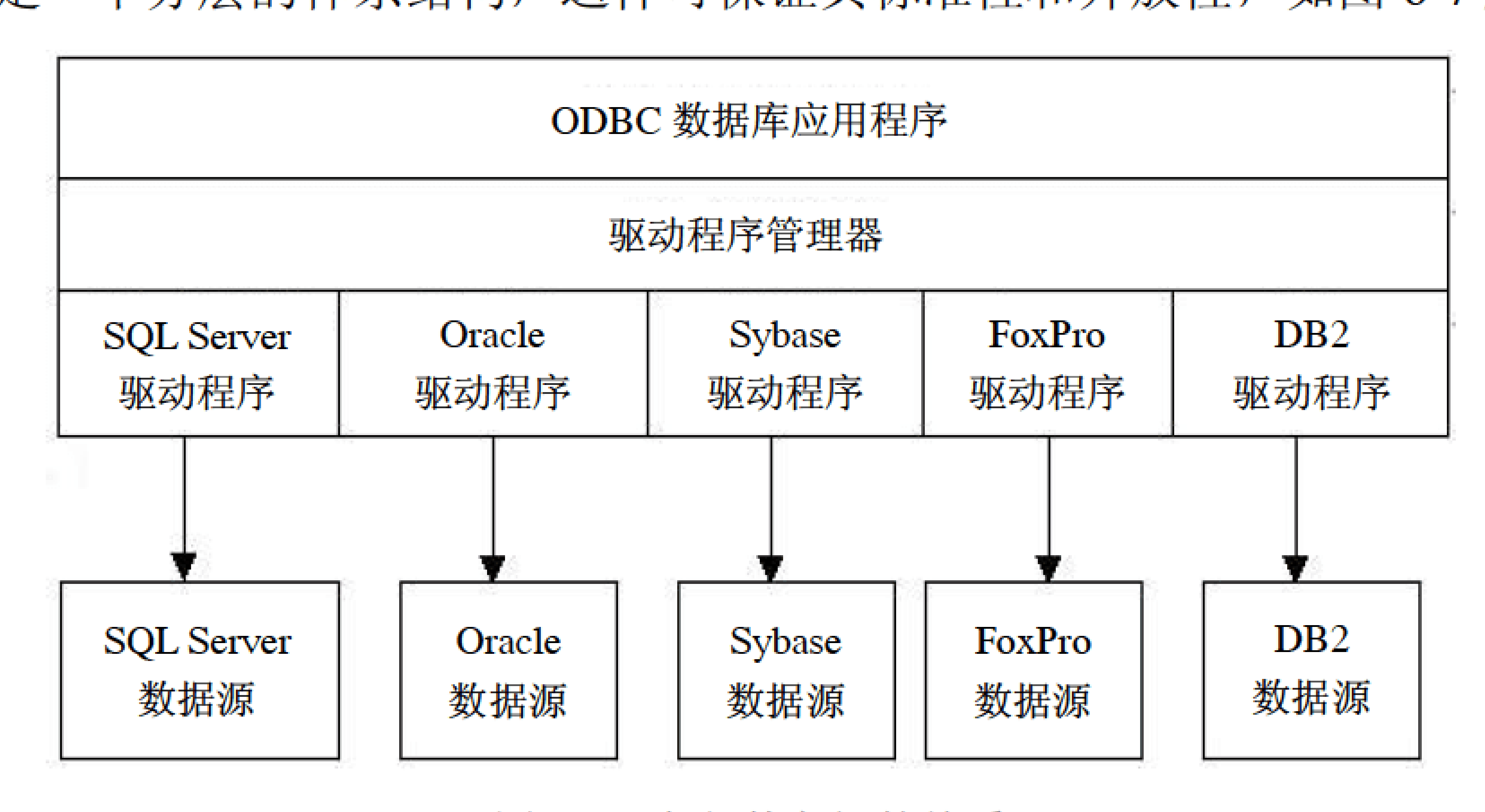


图 6-7　各部件之间的关系

ODBC 由纵向 4 部分构成，其主要功能如下：

（1）ODBC 数据库应用程序（Application）：用宿主语言和 ODBC 函数编写的应用程序用于访问数据库。其主要任务是管理安装的 ODBC 驱动程序和管理数据源。

（2）驱动程序管理器（Driver Manager）：驱动程序管理器包含在 ODBC32.DLL 中，对用户是透明的。其任务是管理 ODBC 驱动程序，为应用程序加载、调用和卸载 DB 驱动程序，是 ODBC 中最重要的部件。

（3）DB 驱动程序（DBMS Driver）：是一些 DLL，提供了 ODBC 和数据库之间的接口。处理 ODBC 函数，向数据源提交用户请求执行的 SQL 语句。

（4）数据源（Data Source）：是 DB 驱动程序与 DBS 之间连接的命名。数据源包含了数据库位置和数据库类型等信息，实际上是一种数据连接的抽象。

微软公司对 ODBC 规程进行了规范，它为应用层的开发者和用户提供标准的函数、语法和错误代码等，微软还提供了驱动程序管理器，它在 Windows 中是一个动态链接库，即 ODBC32.DLL。驱动程序层由微软、DBMS 厂商或第三开发商提供，它必须符合 ODBC 的规程。例如，对于 SQL Server，它的驱动程序是 SQLSRV32.DLL。可以从 ODBC 数据源管理的驱动程序页得到，如图 6-8 所示。



图 6-8　已安装的 ODBC 驱动程序

下面详细介绍各层的功能：

（1）ODBC 数据库应用程序（Application）

使用 ODBC 接口的应用程序可执行以下任务：

① 请求与数据源的连接和会话（SQLConnect）。

② 向数据源发送 SQL 请求（SQLExecDirct 或 SQLExecute）。

③ 对 SQL 请求的结果定义存储区和数据格式。

④ 请求结果。

⑤ 处理错误。

⑥ 如果需要，把结果返回给用户。

⑦ 对事务进行控制，请求执行或回退操作（SQLTransact）。

⑧ 终止对数据源的连接（SQLDisconnect）。

（2）驱动程序管理器（Driver Manager）

应用程序要访问一个数据库，首先必须用 ODBC 管理器注册一个数据源，管理器根据数据源提供的数据库位置、数据库类型及 ODBC 驱动程序等信息，建立起 ODBC 与具体数据库的联系。这样，只要应用程序将数据源名提供给 ODBC，ODBC 就能建立起与相应数据库的连接。这样，应用程序就可以通过驱动程序管理器与数据库交换信息。驱动程序管理器负责将应用程序对 ODBC API 的调用传递给正确的驱动程序，而驱动程序在执行完相应的操作后，将结果通过驱动程序管理器返回给应用程序。

由微软提供的驱动程序管理器是带有输入库的动态连接库 ODBC32.DLL，其主要目的是装入驱动程序，此外还执行以下工作：

① 处理几个 ODBC 初始化调用。

② 为每一个驱动程序提供 ODBC 函数入口点。

③ 为 ODBC 调用提供参数和次序验证。

（3）驱动程序（Driver）

驱动程序是实现 ODBC 函数和数据源交互的 DLL，当应用程序调用 SQL Connect 或者 SQLDriver Connect 函数时，驱动程序管理器装入相应的驱动程序，它对来自应用程序的 ODBC 函数调用进行应答，按照其要求执行以下任务：

① 建立与数据源的连接。

② 向数据源提交请求。

③ 在应用程序需求时，转换数据格式。

④ 返回结果给应用程序。

⑤ 将运行错误格式化为标准代码返回。

⑥ 在需要时说明和处理光标。

以上这些功能都是对应用程序层功能的具体实现。驱动程序的配置方式可以划分为以下两种。

① 单层次（single-tier）：这种方式下，驱动程序要处理 ODBC 调用 SQL 语句，并直接操纵数据库，因此具有数据存取功能。这种配置最常见的是同一台微机之上异种数据库通过 ODBC 存取，如在 PowerBuilder 中存取 Excel、Paradox 等数据文件，如图 6-9 所示。

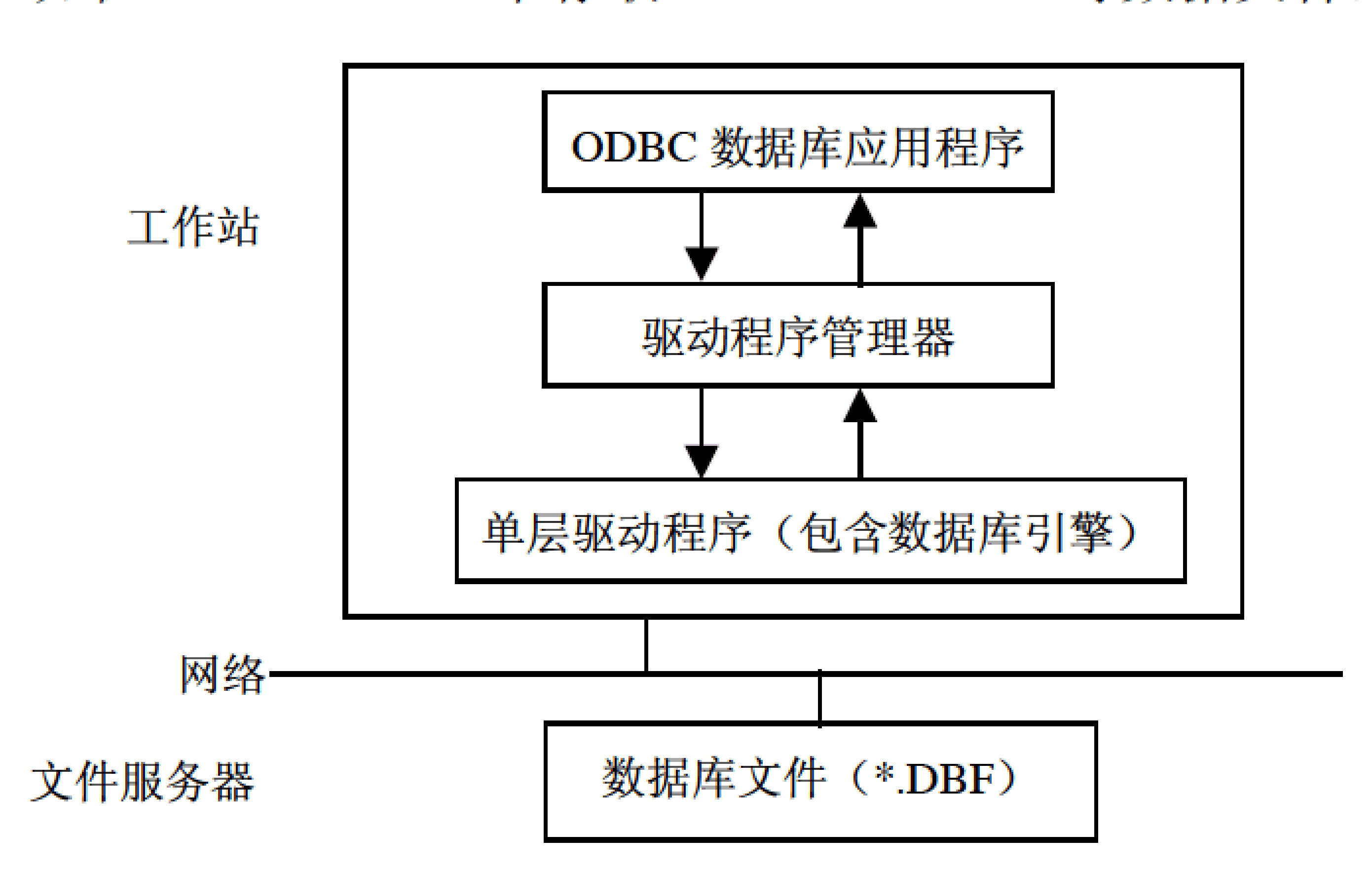


图 6-9　网络环境下基于单层驱动程序的 ODBC 结构

② 多层次（multiple-tier）：这种配置中驱动程序仅处理 ODBC 调用，而将 SQL 语句交给服务器执行，然后返回结果。这种情况往往是应用程序、驱动程序管理器、驱动程序驻留在客户机端，而数据源和数据存取功能放在服务器端。例如，用 FoxPro 或 PowerBuilder 存取 SQL Server 或 Oracle 上的数据，如图 6-10 所示。

有时在以上两者之间加上网关以解决通信协议的转换等问题，这时驱动程序要将请求先传送给网关，如图 6-11 所示。

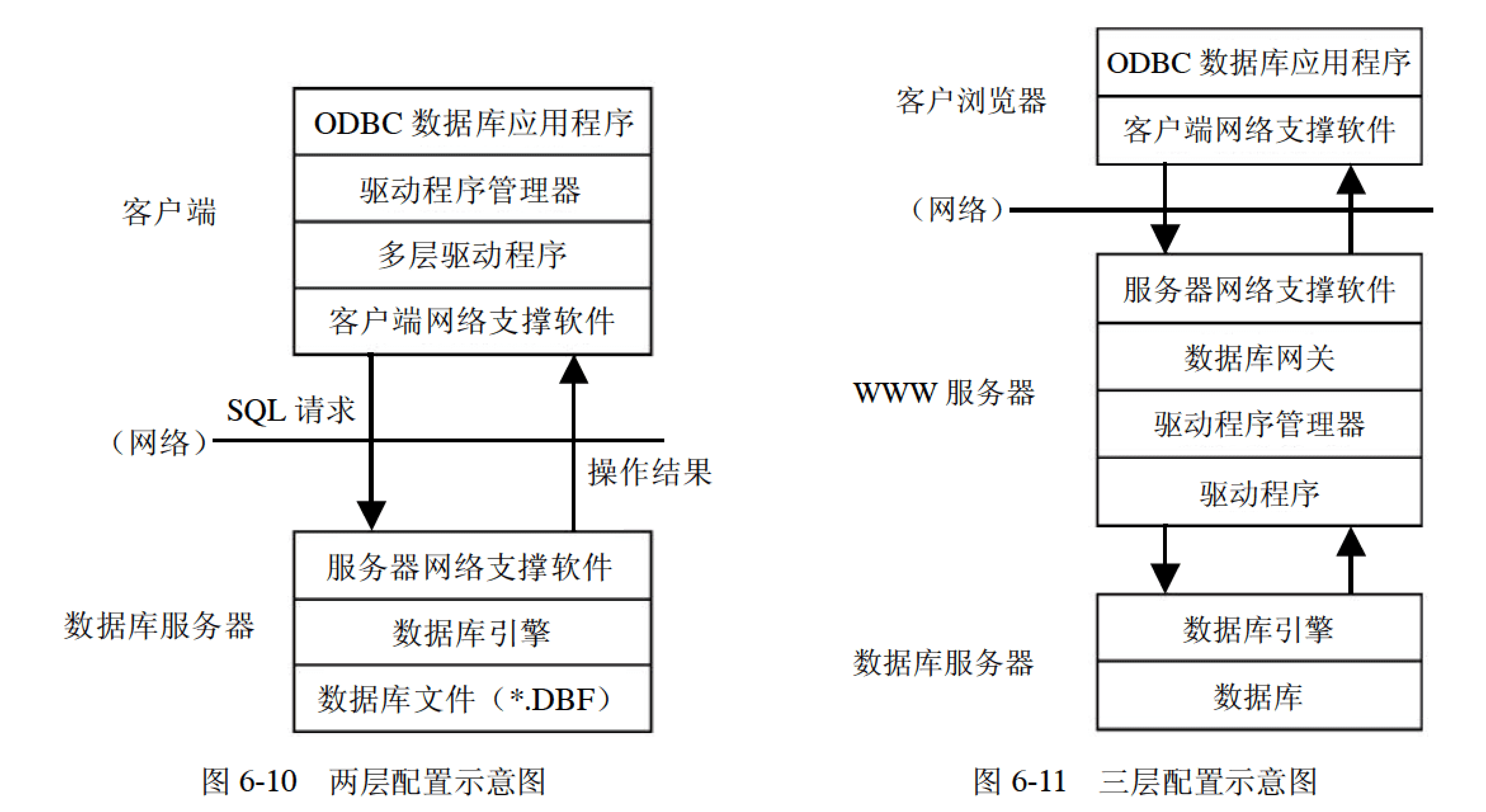


图 6-10　两层配置示意图　　　　图 6-11　三层配置示意图

（4）数据源

数据源（Data Source Name，DSN）是驱动程序与 DBMS 连接的桥梁，数据源不是 DBMS，而是用于表达一个 ODBC 驱动程序和 DBMS 特殊连接的命名。在连接中，用数据源名来代表用户名、服务器名、连接的数据库名等，可以将数据源名看成是与一个具体数据库建立的连接。

数据源由用户想要存取的数据和它相关的操作系统、DBMS 及网络环境组成。数据源分为如下 3 类。

- 用户数据源：ODBC 用户数据源存储了如何与指定数据库提供者连接的信息。只对当前用户可见，而且只能用于当前机器上。这里的当前机器是指这个配置只对当前的机器有效，而不是说只能配置本机上的数据库。它可以配置局域网中另一台机器上的数据库。
- 系统数据源：ODBC 系统数据源存储了如何指定数据库提供者连接的信息。系统数据源对当前机器上的所有用户都是可见的。也就是说在这里配置的数据源，只要是这台机器的用户都可以访问。
- 文件数据源：ODBC 文件数据源允许用户连接数据提供者。文件 DSN 可以由安装了相同驱动程序的用户共享。这是介于用户 DSN 和系统 DSN 之间的一种共享情况。

创建数据源最简单的方法是使用 ODBC 驱动程序管理器，具体的设置后文有介绍。

ODBC 的出现给用户描绘了一个诱人的前景，即网络中的 Windows 用户可以方便地访问各种数据库。现在，在微软推出的许多产品中都提供了 ODBC 支持，同时其他一些应用软件和开发工具也提供了对 ODBC 的支持。因此用户只要安装不同的 ODBC 驱动程序，即可存取相应的数据库产品，而不管用户使用何种前台应用软件，也不管后台是何种数据库，这个存取的过程是一致的。

但是 ODBC 应用存在着一些问题。首先，它的层次比较多，表现在性能上比专有的 API 要慢，这是其标准化和开发性所带来的必要的代价；其次，由于 ODBC 规定了 3 个层次的一致性级别，应用程序与驱动程序之间的匹配就会出现一些问题和矛盾，例如，某些驱动程序支持的级别比较低，而应用程序要求的比较高。

6.2.3　JDBC 与 ODBC

Microsoft 的 ODBC API 可能是使用最广的、用于访问关系数据库的编程接口。它能在几乎所有平台上连接几乎所有的数据库。那么，为什么 Java 不使用 ODBC？答案是显然的：ODBC 不适合直接在 Java 中使用，因为它使用 C 语言接口。从 Java 调用本地 C 代码在安全性、实现、坚固性和程序的自动移植性方面都有许多缺点。从 ODBC C API 到 Java API 的字面翻译是不可取的。例如，Java 没有指针，而 ODBC 却对指针用得很广泛。

因此，Java 语言推出后，为了在 Java 语言中提供对数据库的支持，SUN 公司于 1996 年推出了 JDBC。JDBC 实际上是一组用于执行 SQL 语句的 Java API，由一些 Java 类和接口组成。JDBC API 描述了一套访问关系数据库的标准 Java 类库。可以在程序中使用这些 API，连接到关系数据库，执行 SQL 语句，对数据进行处理。JDBC 建立在 ODBC 上而不是从零开始，它保留了 ODBC 的基本设计特征；事实上，两种接口都基于 X/Open SQL CLI（调用级接口）。它们之间最大的区别在于：JDBC 以 Java 风格与优点为基础并进行优化，因此更加易于使用。可以将 JDBC 想象成被转换为面向对象接口的 ODBC，而面向对象的接口对 Java 程序员来说较易于接收。JDBC 尽量保证简单功能的简便性，而同时在必要时允许使用高级功能。如果完全用 Java 编写 JDBC 驱动程序，则 JDBC 代码在所有 Java 平台上都可以自动安装、移植并保证安全性。

6.2.4　建立 ODBC 数据源

通过 ODBC 去访问 MySQL 的唯一要求是系统安装有 Connector/ODBC，它是专门为 MySQL 数据库而开发的 ODBC 驱动。它的早期版本叫做 MyODBC。如果想确定计算机中是否已经安装了 Connect/ODBC，先通过选择开始→设置→控制面板→管理工具→ODBC 数据源命令，打开“ODBC 数据源管理器”对话框，然后查看“驱动程序”选项卡中是否有 Connector/ODBC 驱动程序。对比图 6-12 和图 6-13。

如果没有 Connector/ODBC 驱动程序就需要安装它，具体安装方法如下：

在 http://dev.mysql.com/downloads/connector/odbc/5.0.html 网页上，由厂商提供了 MySQL 的 JDBC 驱动，单击 Windows MSI Installer 后面的 Download 进行下载，文件名为

mysql-connector-odbc-5.00.11-beta-gpl-win32.msi，安装步骤如图 6-14~图 6-18 所示。



图 6-12　没有安装 Connector/ODBC



图 6-13　安装了 Connector/ODBC

单击 Next 按钮，出现 3 个选项，Typical 为典型安装，能满足基本功能；Complete 为完全安装，需要多一些磁盘空间；Custom 为定制安装，可以自己选择所需要的选项，较为灵活。

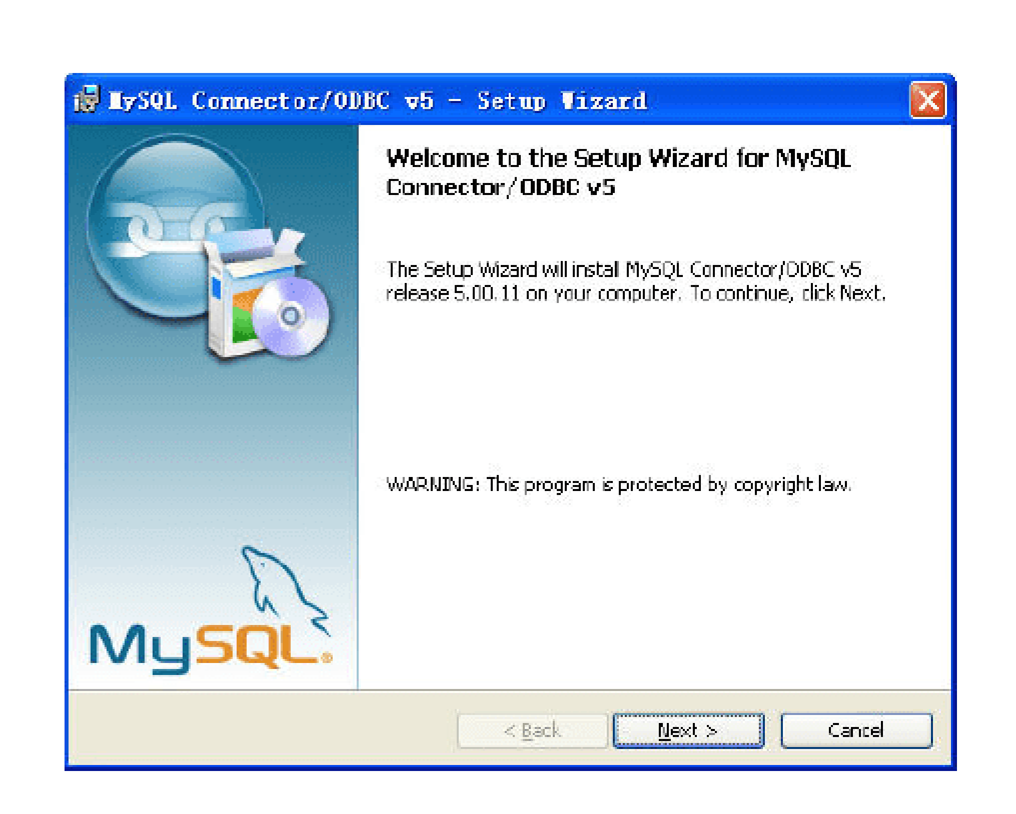


图 6-14　MySQL Connector/ODBC 安装步骤一

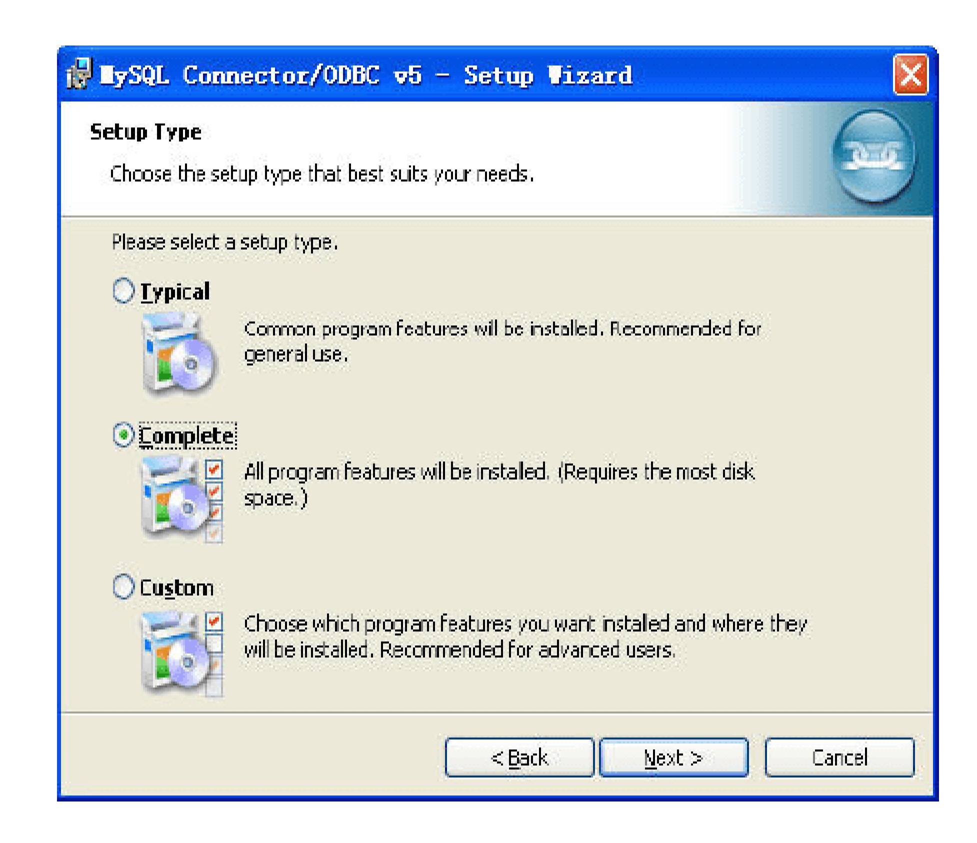


图 6-15　MySQL Connector/ODBC 安装步骤二

在这里，选中 Complete 单选按钮，继续单击 Next 按钮。

然后单击 Install 按钮进行安装。

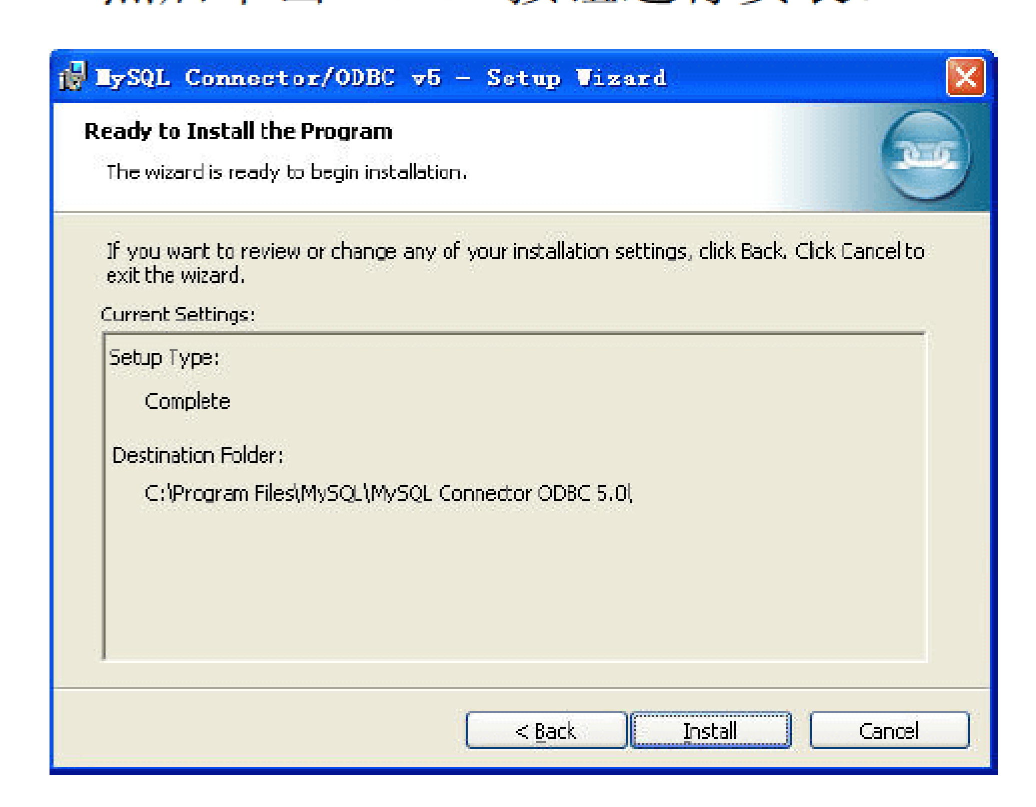


图 6-16　MySQL Connector/ODBC 安装步骤三

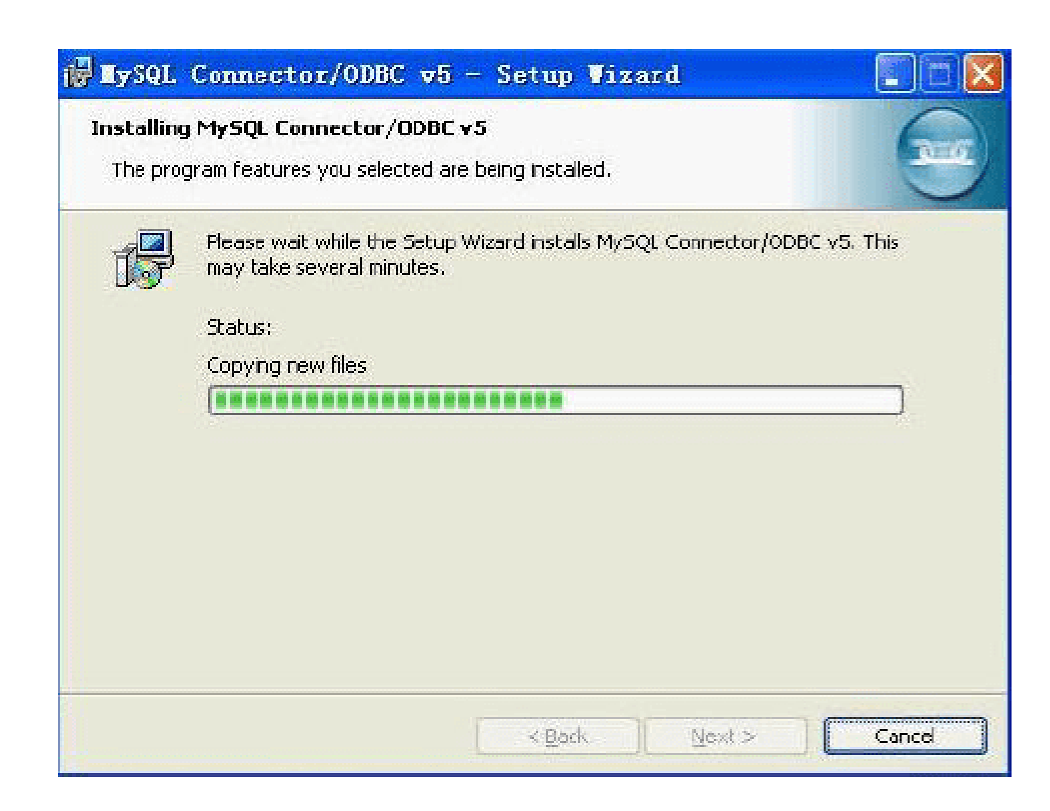


图 6-17　MySQL Connector/ODBC 安装步骤四

单击 Finish 按钮结束。这样，MySQL 的 ODBC 驱动已安装完成。下面来配置 ODBC

数据源。通过选择“开始”→“设置”→“控制面板”→“管理工具”→“ODBC 数据源”命令，打开“ODBC 数据源管理器”对话框，如图 6-19 所示。

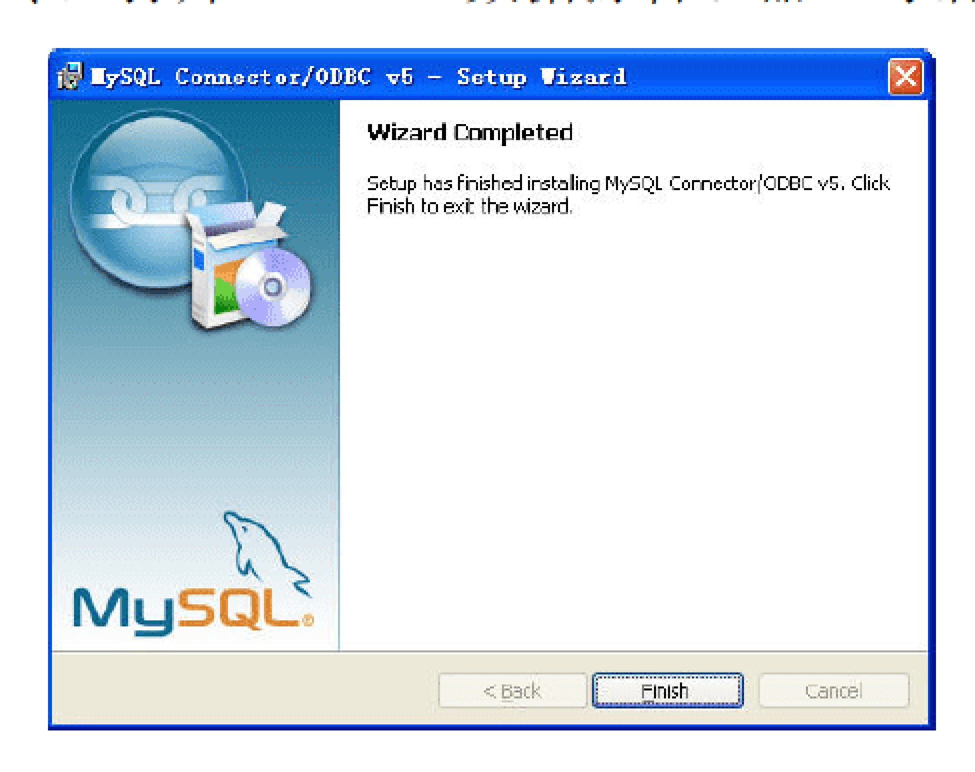


图 6-18　MySQL Connector/ODBC 安装步骤五



图 6-19　“ODBC 数据源管理器”对话框

数据源分为如下 3 类。

- 用户 DSN：只有创建者才能使用并且只能在所定义的机器上运行。
- 系统 DSN：所有用户和 Windows NT 下以服务方式运行的应用程序均可使用系统数据源。用户 DSN 和系统 DSN 的设置数据由 ODBC 内部管理，用户是查看不到的。
- 文件 DSN：文件数据源是 ODBC 3.0 以上版本增加的一种数据源，用于企业用户。这些 DSN 其实是一些*.dsn 文件，它们以文本格式保存着与数据源建立连接所需要的所有设置数据。这些文件通常存放在 Program Files\Common File\ODBC\Data Sources 下，这个设置是可以通过设置目录来改变的，从某种意义上讲，File DSN 相对要更透明一些。要注意的是，对本地数据库来说，通常要在 User DSN（用户 DSN）选项卡上创建一个项；对远程数据库，则在 System DSN（系统 DSN）选项卡上创建。任何情况下，都不能在 User DSN（用户 DSN）和 System DSN（系统 DSN）选项卡上创建同名的项。通常会出现的问题是，如果试图访问远程数据库，就会从 Web 服务器获得非常奇怪和矛盾的错误消息。

下面添加一个文件 DSN，单击“添加”按钮，将弹出如图 6-20 所示的“创建新数据源”对话框。

选择 MySQL Connector/ODBC v5 选项，然后单击“完成”按钮，将打开一个如图 6-21 所示的对话框，需要做一些设置。



图 6-20　“创建新数据源”对话框

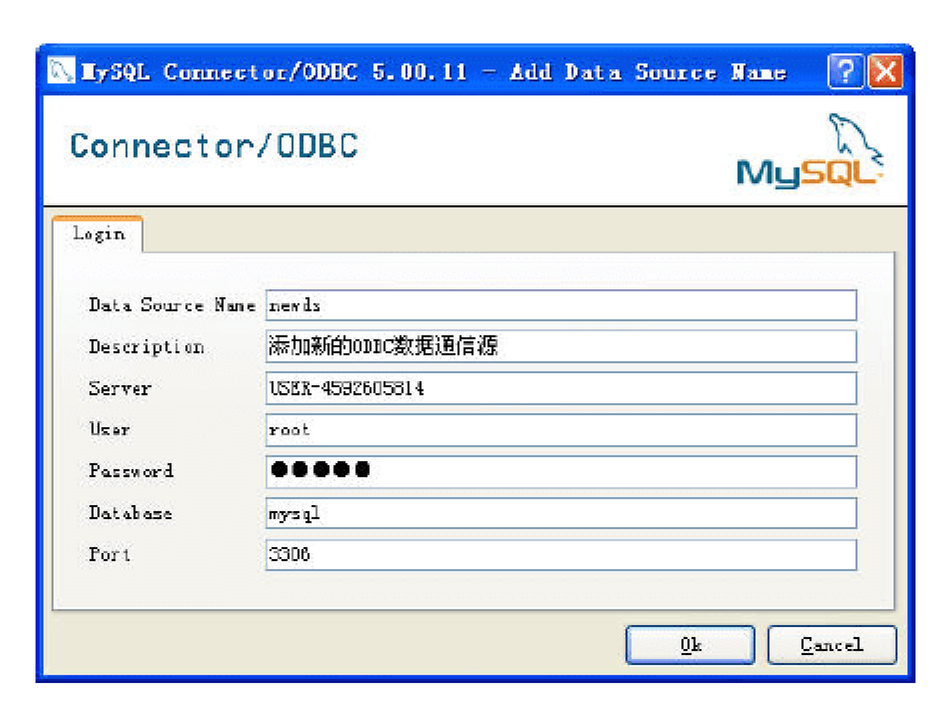


图 6-21　Connector/ODBC 的基本设置

- Data Source Name：命名一个新数据源，ODBC 程序根据这个名字去寻找和访问数据源。注意选择一个有意义并且容易记忆的名字。
- Server：给出 MySQL 服务器的主机名或 IP 地址。
- User：用户名。
- Password：密码。
- Database：MySQL 数据库的名字。
- Port：在安装时设置的，默认值为 3306。

单击 OK 按钮后，就设置好了一个新数据源，如图 6-22 所示。



图 6-22　新添加了一个数据源

如果今后要为数据库更改这些设置，只要简单地加亮它并单击“配置”按钮。删除数据库配置也很容易，只要加亮 DSN 并单击“删除”按钮即可。

6.3　JDBC 数据库设计方法

JDBC 提供了一套 API，以统一的方式访问各种异构的数据库。JDBC 数据库设计具有以下特点：

（1）独立于平台的数据库访问，这是 J2EE 的跨平台机制和 Java 语言的特点决定的。

（2）数据库位置透明，应用程序设计人员不需要关心底层数据库的具体位置、数据库的具体类型等这些不同种类的数据库信息可以通过 JDBC 屏蔽掉，应用程序设计人员可以专注于业务逻辑的实现，只需配置 JDBC URL 即可，并可以方便地实现数据库或应用程序的移植。

（3）对专有数据库的问题是透明的。对于不同的数据库只需要配置不同的 JDBC Driver 以及不同的 JDBC URL，以标准的 JDBC 的方式连接到底层不同技术实现的数据库。

6.3.1　JDBC 的数据库访问模型

Java 的客户端程序大致可分为两类，即 Java Applet 和 Java Application。相对于客户端

来说，JDBC API 支持两种数据库访问模型，即两层模型和三层模型。JDBC 两层应用模型如图 6-23 所示。

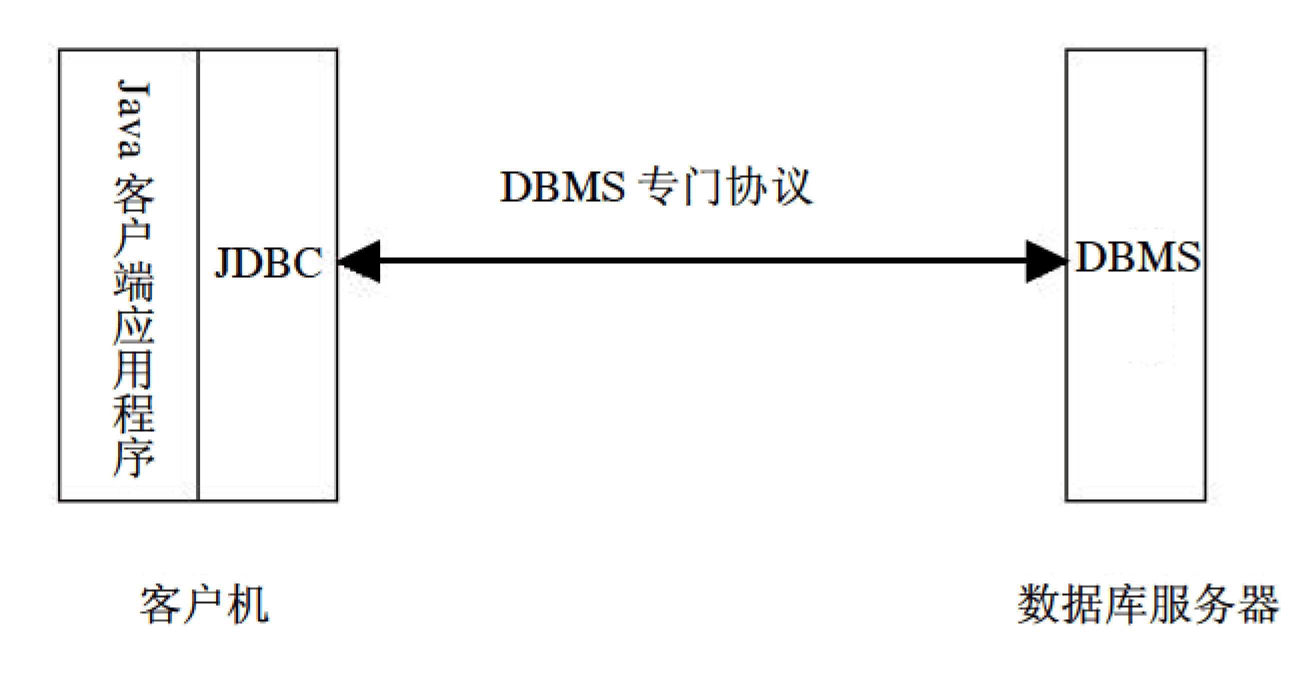


图 6-23　JDBC 两层应用模型

在两层模型中，Java Applet 或 Java Application 将直接与数据库进行对话。其中需要一个 JDBC Driver 来与所访问的特定数据库管理系统进行通信。用户的 SQL 语句被送往数据库中，返回其结果给用户。数据库可以存放在本地机或者是网络服务器上，Java 应用程序也可以通过网络访问远程数据库，如果数据库存放于网络计算机上，则是典型的客户/服务器模型应用。应用程序虽然可以是 Java 的 Application 或 Applet，但是这种模型限制较多，比较适合 Application，而不太适合 Applet。

JDBC 三层应用模型如图 6-24 所示。在三层模型中，客户通过浏览器调用 Java 小应用程序，小应用程序通过 JDBC API 提出 SQL 请求，请求先是被发送到服务的“中间层”，也就是调用小应用程序的 Web 服务器，在服务器端通过 JDBC 与特定数据库服务器上的数据库进行连接，由数据服务器处理该 SQL 语句，并将结果送回到中间层，中间层再将结果送回给用户，用户在浏览器中阅读最终结果。中间层为业务逻辑层，可利用它对公司数据进行访问控制。中间层的另一个好处是，用户可以利用易于使用的高级 API，而中间层将把它转换为相应的低级调用。最后，许多情况下，三层结构可使性能得到优化，并提高安全保证。

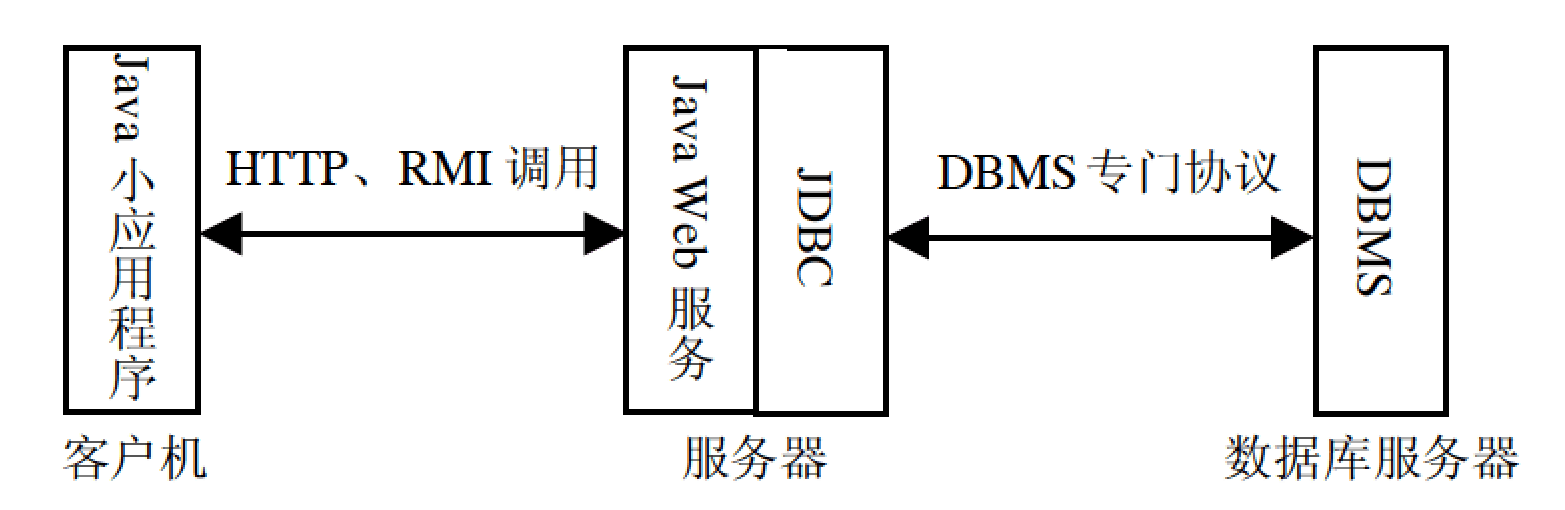


图 6-24　JDBC 三层应用模型

下面是一段两层结构，直接访问数据库的代码示例。

例 6-1　直接访问数据库。

```
import java.sql.*;
public class DatabaseTest {
   public static void main(String[] args)
   {
      try
```

```
      {
         Driver d =(Driver)Class.forName("com.mysql.jdbc.Driver").
         newInstance();
         String url="jdbc:mysql://127.0.0.1:3306/mysql";
         String  user="root";
         String password="admin";
         Connection conn=DriverManager.getConnection(url, user, password);
         Statement stmt=conn.createStatement();
         String query="select * from user";
         ResultSet rs=stmt.executeQuery(query);
         while(rs.next())
            System.out.println(rs.getString(1));
              rs.close();
              stmt.close();
              conn.close();
       }
      catch(Exception e)
      {
          System.out.println("Error:  "+e.toString());
      }
   }
}
```

输出为 localhost。

这是一个具有代表性的数据库访问程序，可以看到，利用 JDBC 访问数据库非常简单，而且有统一的步骤。首先利用 Class 类的 forName()方法，接下来调用 DriverManager 类的 getConnection()方法建立连接，然后调用 Connection 对象的 createStatement()方法，最后调用 Statement 对象的 executeQuery()方法得到 ResultSet 对象。

不管是两层还是三层，都需要 JDBC Driver 的支持，它们与数据库、应用程序的关系如图中所示，在前面曾经提到过 JDBC Driver 的 4 种不同的类型，现在把它们结合到一张图中，如图 6-25 所示可以发现 Type3 其实是一种典型的三层数据模型结构，其中的 Network Server（WLS）就是前面所说的中间层。在 J2EE 架构中，可以通过加载一个 Application Server 来管理 JDBC Driver，对 JDBC Driver 的配置，连接到数据库，以及 JDBC Driver 连接的一些 Connection Pool 配置监控等都可以交给 Application Server 进行统一的管理、监控、配置。同样也可以看出，它与数据库建立连接，事实上还是要借助于 Type1、Type2、Type4。

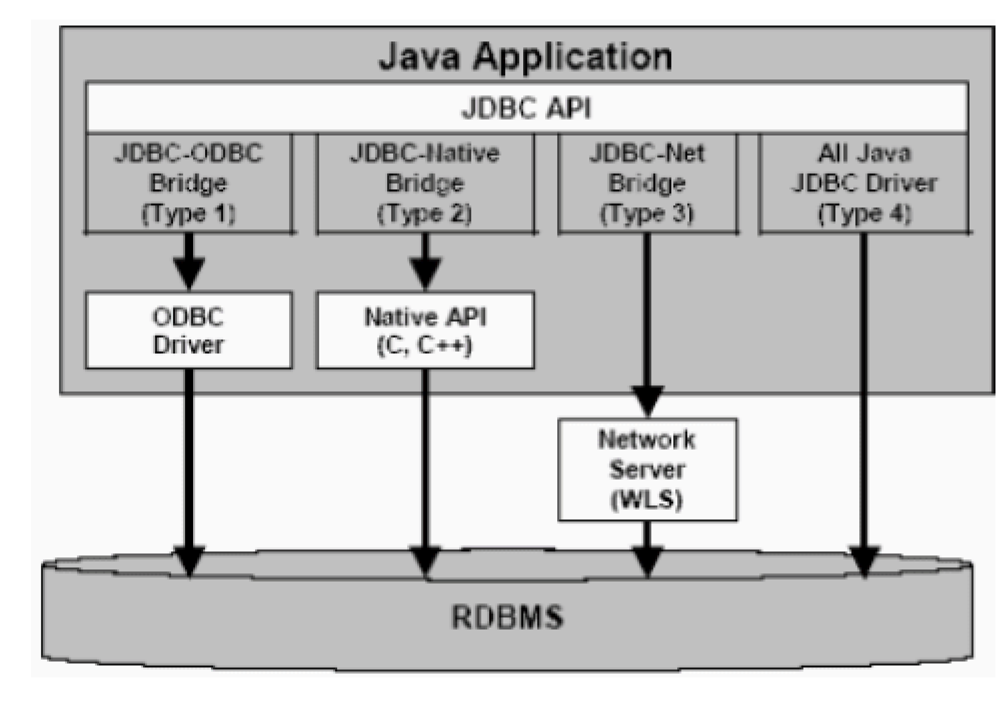


图 6-25　JDBC Driver 的 4 种不同类型

使用三层体系结构，则必须至少配置一个连接池（Connection Pool）和一个数据源（DataSource），如图 6-26 所示。

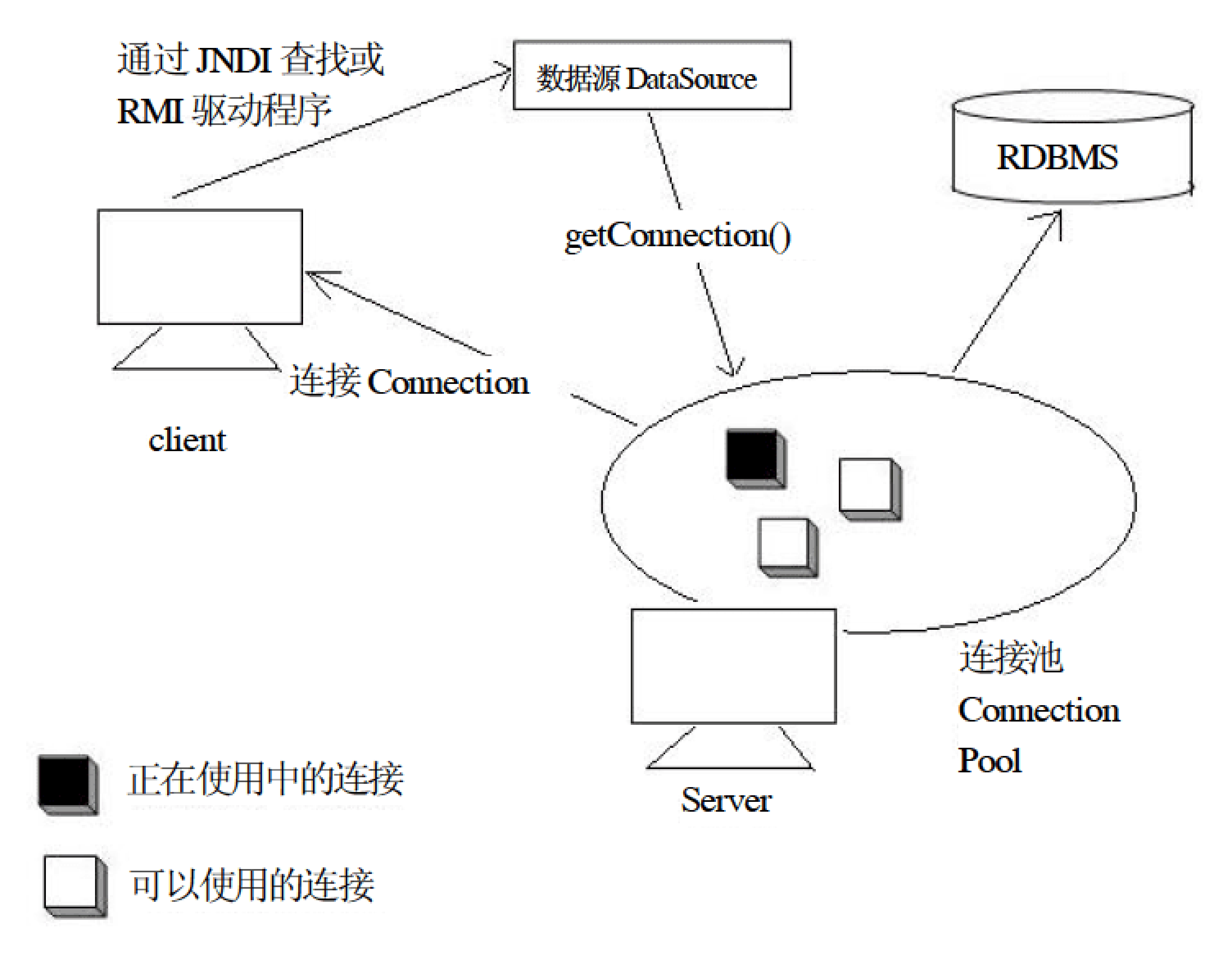


图 6-26　数据池连接

6.3.2　连接池

数据库连接是一种关键的、有限的、昂贵的资源，这一点在多用户的网页应用程序中体现得尤为突出。对数据库连接的管理能显著影响到整个应用程序的伸缩性和健壮性，影响到程序的性能指标。数据库连接池正是针对这个问题提出来的。

数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不是重新建立一个；释放空闲时间超过最大空闲时间的数据库连接来避免因为没有释放数据库连接而引起的数据库连接遗漏。这项技术能明显提高对数据库操作的性能。

数据库连接池在初始化时将创建一定数量的数据库连接放到连接池中，这些数据库连接的数量是由最小数据库连接数来设定的。无论这些数据库连接是否被使用，连接池都将一直保证至少拥有这么多的连接数量。连接池的最大数据库连接数量限定了这个连接池能占有的最大连接数，当应用程序向连接池请求的连接数超过最大连接数量时，这些请求将被加入到等待队列中。数据库连接池的最小连接数和最大连接数的设置要考虑到如下 3 个因素。

（1）最小连接数是连接池一直保持的数据库连接，所以如果应用程序对数据库连接的使用量不大，将会有大量的数据库连接资源被浪费。

（2）最大连接数是连接池能申请的最大连接数，如果数据库连接请求超过此数，后面的数据库连接请求将被加入到等待队列中，这会影响之后的数据库操作。

（3）如果最小连接数与最大连接数相差太大，那么最先的连接请求将会获利，之后超

过最小连接数量的连接请求等价于建立一个新的数据库连接。不过，这些大于最小连接数的数据库连接在使用完不会马上被释放，它将被放到连接池中等待重复使用或是空闲超时后被释放。

多层体系结构的优势表现在如下 3 个方面：

（1）利用连接池可以消除频繁建立连接所需要的负载。

（2）是用于管理数据库连接的管理对象，是在服务器端进行统一的配置、管理和监控的，这既节省了客户端开发的时间和精力，同时有效地保护了后端资源的安全性。

（3）提供可共享的安全连接，如图 6-27 所示。

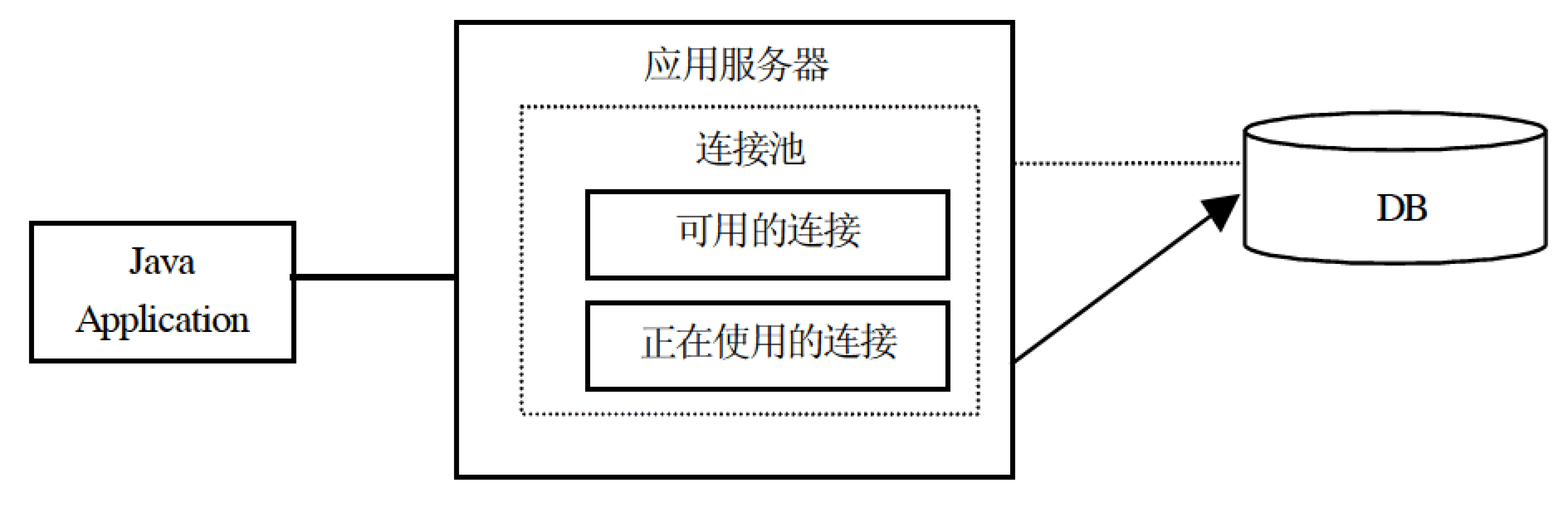


图 6-27　可共享的安全连接

6.4　JDBC 安全性

随着互联网在人们生活中的应用日益广泛深入，并且互联网所具有的资源共享性，因此能够按照用户需求及时准确获得信息和处理信息，对用户而言相当重要，这也是 Java 得以迅速发展和被广泛接受的重要原因。

但同时网络也提供了一条攻击接入计算机的潜在途径，特别是当用户下载网络软件在本地运行，这就要求 Java 能够对病毒/木马的问题加以防范，对信息以及本地环境进行保护。例如浏览一个网页时，网页上的 Applet 可能会自动下载并且运行，而这个 Applet 完全有可能来自不可靠的地方，又或者使用通过 JNDI 服务查找到的网络上不可靠的服务对象来获得服务，这就很有可能引入了一个怀有敌意的程序造成信息丢失、资料泄密、信任伪造数据和修改本地计算机安全设置等各种各样严重的后果。

6.4.1　Java 体系结构对信息安全的支持

Java 体系结构对安全性的支持主要是通过 Java 语言本身安全性、虚拟机的类加载器和安全管理器以及 Java 提供的安全 API 几个方面来实现：防止恶意程序的攻击，程序不能破坏用户计算机环境；防止入侵，程序不能获取主机或所在内网的保密信息；鉴别，验证程序提供者和使用者的身份；加密，对传输交换的数据进行加密，或者给持久化的数据进行加密；验证，对操作设置规则并且进行验证。

为了防止用户系统受到通过网络下载的不安全程序的破坏，Java 提供了一个自定义的

可以在里面运行 Java 程序的“沙盒”。这就是 Java 的安全模型，Java 的安全模型使得 Java 成为适合于网络环境的技术。Java 的安全性允许用户从 Internet 或 Intranet 上引入或运行 Applet，Applet 的行动被限制于它的“沙盒”，Applet 可以在沙盒里做任何事情，但不能读或修改沙盒外的任何数据，沙盒可以禁止不安全程序的很多活动，如对硬盘进行读写，和别的主机（不包括程序所在的主机）进行网络连接，创建一个新过程，载入一个新的动态库并直接调用本地方法。

“沙盒”模型的思想是在信任的环境中运行不信任的代码，这样即使用户不小心引入了不安全的 Applet，Applet 也不会对系统造成破坏。“沙盒”安全模型是内建于 Java 结构的，它主要由以下 4 个部分构成。

（1）内建于 Java 虚拟机和语言的安全特性

Java 取消了多重继承而采用实现多个接口的方式，去除了 C++语言中的令人费解、容易出错的“指针”，用列表、堆、哈希表等结构来代替，避免了任何不安全的结构。结构、单元、运算符重载、虚拟基础类等在 Java 中都没有采用。这样能降低开发人员犯错误的几率，帮助他们写出更安全的代码。

Java 分配内存对于开发人员来说是透明的，开发人员使用 new 方法新建对象，这时虚拟机就会从堆内存中找到合适的内存空间，开发人员不需要也不能够进行干预。而对于内存的回收，Java 避免了开发人员明确干预对象的回收，避免了开发人员无意间对内存的破坏。

对于在网络中交换的序列化对象很容易在重建对象时访问到对象的私有信息，这时 Java 提供了两种办法来保护信息：一种就是采用给变量加上 transient 关键字的方法，这样对象序列化时就不会读写该变量；另一种就是实现 Externalizable 接口而不是 Serizlizable 接口，这样对象就只能通过 writeExternal 和 readExternal 方法来保存和重建，其他方法就无法进行。

以上这些都是 Java 语言本身对信息安全提供的基础。这些机制也是 Java 虚拟机（JVM）的特点。

（2）类的载入结构（类加载器）

类载入器结构在沙盒模型中起了重要作用。在虚拟机中，类载入器负责引入、定义运行程序的类和接口的二进制数据。在虚拟机中可能有不止一个类载入器。

类加载器避免了出现某些怀有敌意的人编写自己的 Java 类，而这些类方法中含有跳转到方法之外的指令，导致虚拟机的崩溃和保密信息被获取的可能。类加载器保证了程序的健壮性，也不会出现替代原有 Java API 类的恶意代码被运行的情况，并且类加载器防止了恶意代码去干涉善意的代码，守护了被信任的 API 类库边界，确保了代码可以进行的操作。

在“沙盒”结构中，类载入器结构是防止不安全代码的第一道围墙，它的作用主要有两方面：

① 防止不安全代码访问、破坏安全代码。

② 防止不安全代码冒充安全的类。

（3）类文件校验器

每一个 JVM 都有一个类文件校验器，用来保证载入的类文件具有正确的内部结构。如果类文件校验器发现类文件有错误，它就抛出一个异常。

类文件校验器能帮助检查出类使用起来是否安全。因为类文件是由二进制数据构成的，

JVM 不知道这个类文件是否是由黑客产生的，是否有可能破坏虚拟机的完整性，所以虚拟机对引入的字节码进行校验是很重要的。类文件校验器校验的过程可以分为两个阶段。

- 阶段一：发生在类刚载入以后，类文件校验器检查类文件的内部结构，检查类文件是否正确组成、内部是否一致、是否遵循 Java 编程语言的限制、含有的字节码是否能由 JVM 安全执行，如果类文件校验器检查出错误，它就会抛出一个错误，类文件就不再被程序使用，包括校验所含的字节码的完整性。
- 阶段二：发生在字节码执行时，字节码校验器确定符号引用的类、域和方法是否存在内部检查，通过对代表类方法的字节码流进行数据流分析，进行操作码是否有效及操作码是否有有效的操作数等的检查，以验证字节码流是否可以由虚拟机安全执行。

（4）安全管理器和 Java API

安全管理器定义了“沙盒”的外部边界。安全管理器是类 java.lang.SecurityManager 的子类，它是自定义的。

Java API 类在采取一些行动时，通常需要安全管理器检查这个行动是否安全，这些行动包括：

- 接受来自于特定主机的 socket 连接。
- 修改线程（改变线程优先级、结束线程等）。
- 开放对于特定主机的 socket 连接。
- 创建一个新的类载入器。
- 删除特定的文件。
- 创建新的过程。
- 程序退出。
- 调用含有本地方法的动态库。
- 等待连接。
- 从特定的包载入类。
- 给特定的包中添加一个新类。
- 访问或修改系统特性。
- 访问特定的系统特性。
- 读文件。
- 写文件。

由于在执行上述动作前需要安全管理器进行检查，Java API 不执行安全管理器建立的安全措施所禁止的任何动作。

6.4.2 JDBC 安全模式

JDBC 是建立在 Java 语言这种强大的安全机制下的，从客户端层面来说，使用 JDBC 可以开发 Java 应用程序和 Java 小应用程序。两种不同的方式体现不同的安全性。对于 Application 和 Applet 必须分别对待。在 JDBC 安全方面需要注意以下问题：

（1）Driver 的安全责任

JDBC Driver 可能在各种情况下使用，所以驱动的编制者遵循一定的、简单的安全规则，从而避免 Applet 做非法的数据库连接。如果所有的驱动都像 Applet 一样从网上下载，那么这些原则将是不必要的，因为普通的安全规则已经对它做了限制。但是驱动的编写者必须记住：一旦他们的驱动获得成功，用户将在本地磁盘安装这些驱动，那么驱动将成为 Java 环境中一个被信任的部分，所以必须确信它不会被来访的 Applet 所滥用。所以我们鼓励所有的驱动编写者必须遵循一定的安全原则。所有这些原则都是在连接打开时使用，一旦连接建立就不必做更多的检查了。

（2）分享 TCP/IP 连接时必须谨慎

如果一个 JDBC 驱动试图打开一个 TCP 连接，那么这个打开会被 Java 安全管理机制自动检查。这个机构会检查当前调用栈里面有没有 Applet，如果有那么就限定它可以访问的机器集合。所以一般的 JDBC 驱动可以把 TCP 建立检查留给 Java 虚拟机。但是如果一个 JDBC 驱动试图在多个数据库连接之间共享一个 TCP 连接，那么驱动就必须自己负责检查每个调用者是否真的被允许与目标数据库联系。例如，如果为 Applet A 打开了一个 TCP 连接，这并不意味着 Applet B 被自动允许来共享这个连接。Applet B 可能没有任何访问的权力。所以在允许某个程序重用一个现成的 TCP 连接之前，JDBC 驱动必须通过安全机构来检查当前的调用者是否可以访问这个连接。通过下面的代码可以实现这个功能：

```
SecurityManager security = System.getSecurityManager();
if (security != null)
{
    security.checkConnect(hostName, portNumber);
}
```

如果连接是不允许的，那么 Security.checkConnect 方法将产生一个 java.lang.Security Exception 异常。

（3）做好最坏的准备

一些驱动可能使用本地的方法来桥接底层数据库程序，则这些情况里面判断哪些本地文件将被底层函数所访问是困难的。

在这些环境里面用户必须做好最坏的打算，并且否决所有下载 Applet 所发出的数据库存取，除非驱动可能完全确信将要做存取是没有问题的。

6.5　获取和安装 JDBC

从 Java jdk1.1 版本开始已经包含了 JDBC，因此不需要单独安装 JDBC。

6.6　关于 JDBC API

JDBC API 是一种成熟的技术，主要目标有两个：一是提高所有开发者在 Java 平台使用

SQL 开发的易用性，二是提供企业级特性的 JDBC 工具集和 API 来管理 JDBC 资源。

6.6.1　接口概貌

JDBC 接口分为两个层次，一个是面向程序开发人员的 JDBC API。另外一个是底层的 JDBC Driver API，驱动程序层是驱动厂家实现的。每一个驱动程序层都必须实现 4 个主要的接口，应用程序层和驱动程序层用一个类桥连接。这 4 个接口分别是 Driver、Connection、Statement 和 ResultSet。图 6-28 表达了 JDBC Driver API 和底层数据库的连接，其中包括 JDBC-ODBC 桥驱动和一些典型的驱动程序。

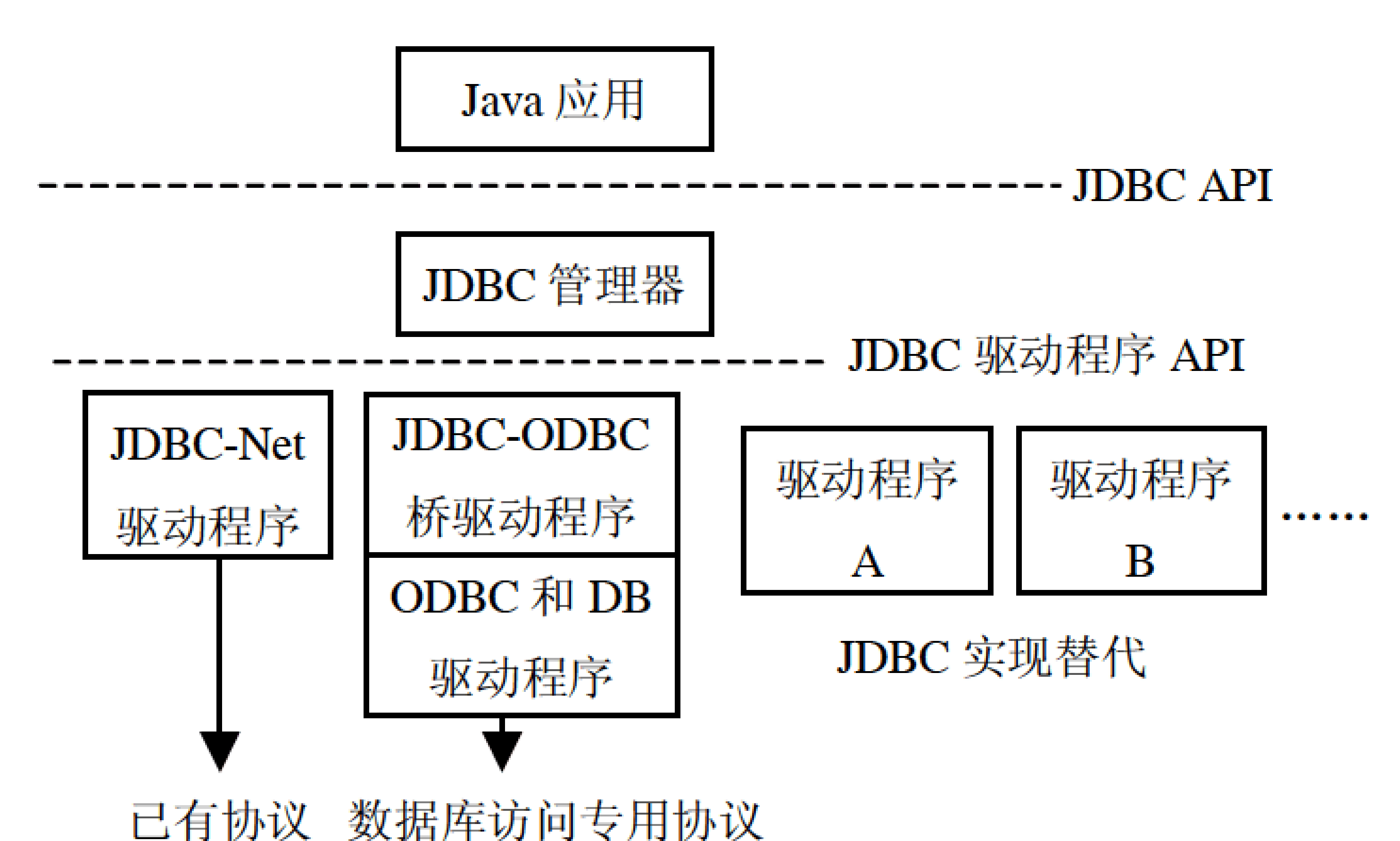


图 6-28　JDBC Driver API 和底层数据库的连接

应用程序层与驱动程序层的连接如图 6-29 所示。

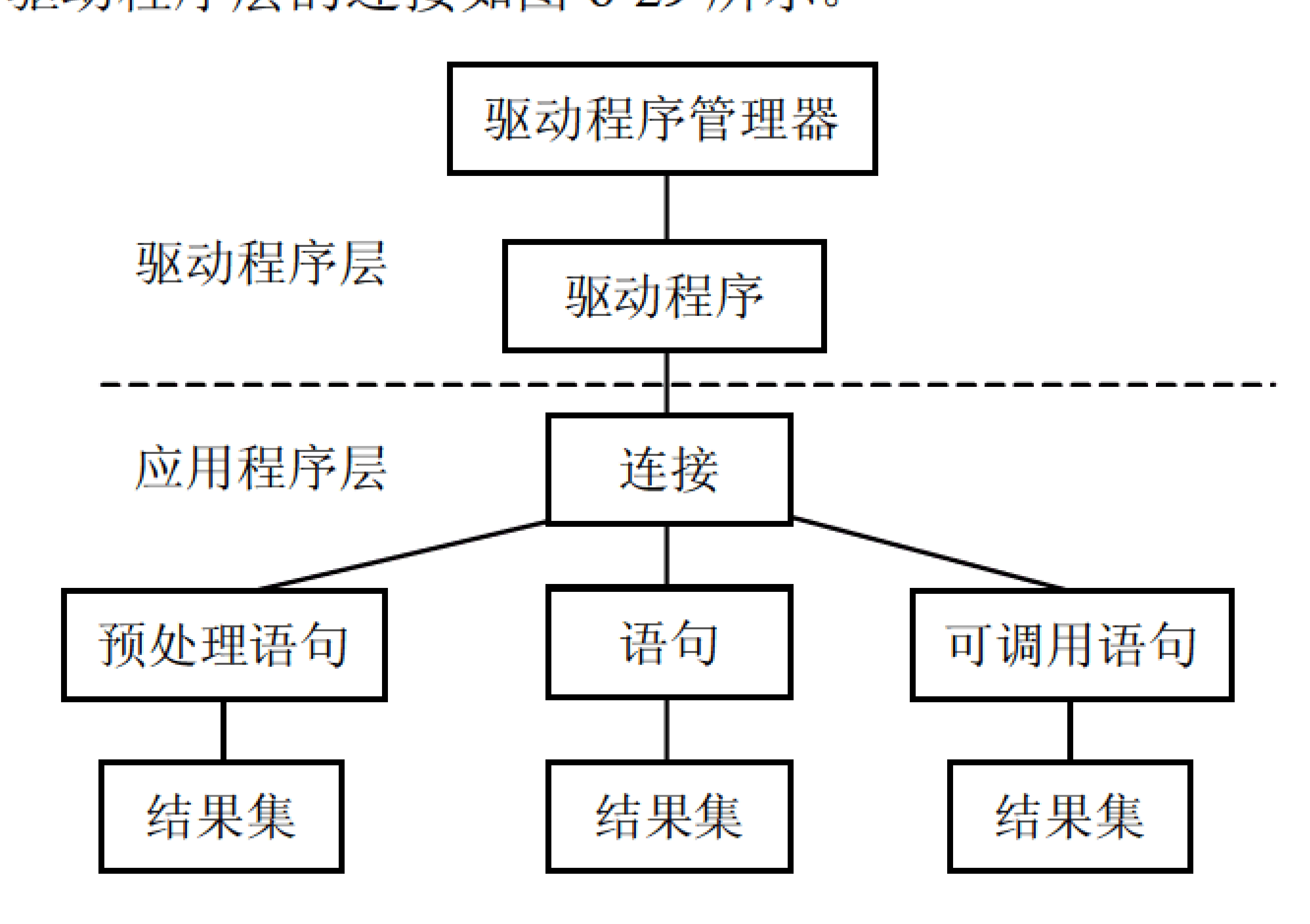


图 6-29　应用程序层与驱动程序层的连接

应用程序层需要完成的 3 个接口是在驱动程序层实现的。Java 接口提供了用一般名称表示具体对象的方法。对于应用程序开发者，具体 Driver 类实现并不重要，只要编码符合 JDBC API 标准就足够了。

JDBC API 被描述成一组抽象的 Java 接口，应用程序可以对某个数据库建立连接，执行

SQL 语句并处理结果。3 个主要的接口是 Connection、Statement 和 ResultSet，图 6-30 表达了这 3 个接口的关系。

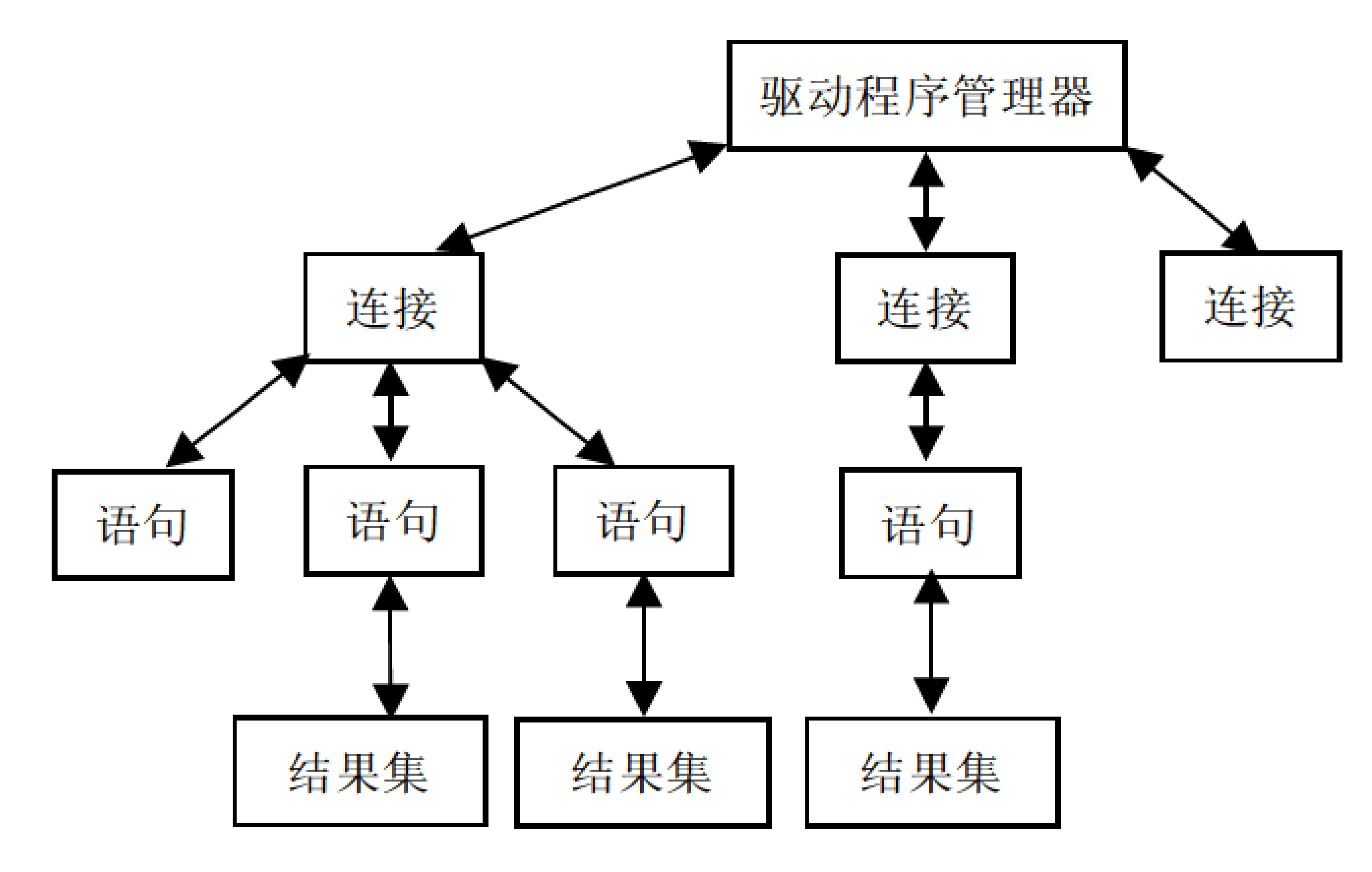


图 6-30　3 个接口的关系

在数据库的三层或者多层设计中，数据库和 JDBC 驱动程序之间都是一一对应的。Driver 类是驱动呈现厂家实现的接口。另一类是 DriverManager 类，在驱动程序和应用程序层之上。DriverManager 主要是负责装入和拆除驱动程序及联机驱动程序。

6.6.2　JDBC API 的接口和类

JDBC API 包含在 JDK 中，被分为两个包：java.sql 和 javax.sql。java.sql 包定义了访问数据库的接口和类，其中一些接口由驱动程序提供商来实现。javax.sql 补充了 java.sql 包，它从 1.4 版本开始包含在 JDK 中。它保留了 Java 2 SDK 企业版中的精华部分。

java.sql 包提供的接口介绍如下。

（1）java.sql.Array

在 Java 语言中，该接口表示数据库 Array 类型的映射。Array 对象是一个逻辑指针，它指向服务器上的 SQL Array 值中的数据。

（2）java.sql.Blob

在 Java 语言中，该接口表示 SQL 二进制的对象（Blob）数据类型。Blob 对象包含指向数据的逻辑指针，而不包含数据自身。

（3）java.sql.CallableStatement

CallableStatement 用于执行 SQL 存储过程的接口。

JDBC 提供了一个存储过程 SQL escape，它允许以所有 RDBMS 的标准方式调用存储过程。该 escape 语法具有一个包含结果参数的格式和一个不包含结果参数的格式。如果使用结果参数，它必须被注册为一个 OUT 参数。其他的参数可以用于输入、输出或同时用于两者。参数通过编号被顺序地引用。第一个参数是 1。

用从 PreparedStatement 继承的设置方法设置 IN 参数。所有 OUT 参数的类型必须在执行该存储过程之前注册，执行后的参数值用该类提供的 get 方法获得。

Callable 语句可能返回一个 ResultSet 或多个 ResultSet。多个 ResultSets 由从 Statement 继承的操作处理。

为了最大限度的可移植性，调用的 ResultSet 和修改次数必须在获得输出参数值之前处理。

（4）java.sql.Clob

该接口表示 SQL Character Large Object（Clob）数据类型。Clob 对象包含指向数据的逻辑指针，而不包含数据自身。

（5）java.sql.Connection

该接口表示与特定数据库的连接（会话）。

一个 Connection 表示与一个特定数据库的会话。在一个 Connection 的上下文中，执行 SQL 语句并返回结果。

一个 Connection 的数据库能够提供描述表信息、支持的 SQL 语法、存储过程、连接的能力等内容的信息，该信息可用 getMetaData 方法获得。

注意，默认情况下，在执行完每一个语句之后，Connection 自动地提交更改。如果禁止自动提交，必须进行显式的提交，否则将不保存对数据库的更改。

（6）java.sql.DatabaseMetaData

该接口表示关于数据库的整体综合信息。这里的许多方法将在 ResultSets 中返回信息列表。可以使用一般的 ResultSet 方法，如 getString 和 getInt，从这些 ResultSet 中检索数据。如果给定的元数据格式是不可用的，这些方法将抛出一个 SQLException。

（7）java.sql.Driver

表示每个驱动程序类必须实现的接口。

（8）java.sql.ParameterMetaData

该接口可用于获取关于 PreparedStatement 对象中参数的类型和属性信息的对象。

（9）java.sql.PreparedStatement

该接口表示预编译的 SQL 语句的对象。该对象可用于有效地多次执行该语句。

注意，用于设置 IN 参数值的 setXXX 方法必须指定与定义的输入参数的 SQL type 兼容的类型。例如，如果 IN 参数有 SQL type Integer，则应使用 setInt 方法。如果需要任意的参数类型转换，则 setObject 方法应当与目标 SQL type 一起使用。

（10）java.sql.Ref

该接口表示 Java 编程语言中 SQL REF 值的映射关系，它是到数据库中的 SQL 结构类型值的引用。

（11）java.sql.ResultSet

该接口表示数据库结果集的数据表，通常通过执行查询数据库的语句生成。ResultSet 提供了通过执行一条语句访问所生成的数据表的功能。按顺序获取表中的行。在一行内它的列值可以以任意顺序访问。

ResultSet 控制一个指向当前数据行的游标。初始，游标定位于第一行之前。用 next 方法可把游标移到下一行。

getXXX 方法获取当前行的列值。可通过使用列的索引或名字来获取值。通常使用列索

引会更有效。列索引从 1 开始。

最大概率下，每行的 ResultSet 列应按照从左到右的顺序获取，并且每列只读一次。

对 getXXX 方法，JDBC 驱动程序试图把基础数据转换为指定的 Java 类型并返回一个合适的 Java 值。参见关于使用 ResultSet.getXXX 方法从 SQL 类型到 Java 类型进行映射的 JDBC 规范。

输入到 getXXX 方法的列名是大小写敏感的。当使用列名执行一个 getXXX 方法时，如果几个列有同样的名字，则返回第一个匹配的列。当列名用于 SQL 查询时，指定使用列名选项。对在查询中没有明确命名的列，最好使用列编号。如果使用列名，则无法对编程者保证他们实际使用了所想要的列。

ResultSet 由生成它的语句自动关闭、再执行或从多个结果的序列中获取下一个结果。

ResultSet 列的编号、类型和特性由 getMetaData 方法返回的 ResulSetMetaData 对象提供。

（12）java.sql.ResultSetMetaData

该接口可用于获取关于 ResultSet 对象中列的类型和属性信息的对象。

（13）java.sql.Savepoint

该接口是保存点的表示形式，保存点是可以从 Connection.rollback 方法引用的当前事务中的点。

（14）java.sql.SQLData

该接口用于 SQL 用户定义类型（UDT）到 Java 编程语言中类的自定义映射关系。

（15）java.sql.SQLInput

该接口是一个输入流，它包含表示 SQL 结构化类型或 SQL 不同类型的实例的值组成的流。

（16）java.sql.SQLOutput

该接口用于将用户定义类型的属性写回数据库的输出流。

（17）java.sql.Statement

该接口用于执行静态 SQL 语句并返回它所生成结果的对象。

（18）java.sql. Struct

该接口用于 SQL 结构化类型的 Java 编程语言中的标准映射关系。

java.sql 包提供的类介绍如下。

（1）java.sql.Date

一个包装了毫秒值的瘦包装器（thin wrapper），它允许 JDBC 将毫秒值标识为 SQL DATE 值。

（2）java.sql.DriverManager

DriverManager 类是 JDBC 的管理层，作用于用户和驱动程序之间。它跟踪可用的驱动程序，并在数据库和相应驱动程序之间建立连接。另外，DriverManager 类也处理诸如驱动程序登录时间限制及登录和跟踪消息的显示等事务。

对于简单的应用程序，一般程序员需要在此类中直接使用的唯一方法是 DriverManager. getConnection。正如名称所示，该方法将建立与数据库的连接。JDBC 允许用户调用 DriverManager 的方法 getDriver、getDrivers 和 registerDriver 及 Driver 的方法 connect。但多

数情况下，让 DriverManager 类管理建立连接的细节为上策。

通过调用方法 Class.forName。这将显式地加载驱动程序类。由于这与外部设置无关，因此推荐使用这种加载驱动程序的方法。以下代码加载类 acme.db.Driver：

```
Class.forName("acme.db.Driver");
```

（3）java.sql.DriverPropertyInfo

该类用于建立连接的驱动程序属性。

（4）java.sql.SQLPermission

SecurityManager 将用来检查在 Applet 中运行的代码何时调用 DriverManager.setLogWriter 方法或 DriverManager.setLogStream（不建议使用）方法的权限。

（5）java.sql.Time

该类是一个与 java.util.Date 类有关的瘦包装器（thin wrapper），它允许 JDBC 将该类标识为 SQL TIME 值。

（6）java.sql.Timestamp

该类是一个与 java.util.Date 类有关的瘦包装器（thin wrapper），它允许 JDBC API 将该类标识为 SQL TIMESTAMP 值。

（7）java.sql.Types

该类定义用于标识一般 SQL 类型（称为 JDBC 类型）的常量的类。

java.sql 包提供的异常类介绍如下。

（1）java.sql.BatchUpdateException

当在进行批量更新操作期间发生错误时，抛出该异常。

（2）java.sql.DataTruncation

报告 DataTruncation 警告的异常（读取时），或者 JDBC 意外地截断某一数据值时抛出 DataTruncation 异常（写入时）。

（3）java.sql.SQLException

该异常提供关于数据库访问错误或其他错误的信息。一个描述错误的一个字符串。它被用作 Java Exception 消息，并可通过调用 getMessage()方法使用它。

（4）java.sql.SQLWarning

提供关于数据库访问警告信息的异常。

6.6.3　如何实现 JDBC 性能优化

JDBC 程序的性能主要由两个因素决定，一是数据库本身的性质，另一个是与数据库相对独立的 JDBC 应用程序接口（API）的使用。这里说的是如何正确使用 JDBC 编程接口，以获得更好的性能。

JDBC 的主要优化介绍如下。

（1）选择正确的 jdbc 驱动程序

最好选择 JDBC 网路协议-纯 Java 驱动，效率比较高，但需要第三方软件的支持，例如

corba weblogic 属于这种类型。

（2）优化 Connection 对象

❑ 设置适当的参数

DriverManager.getConnection(String url,Properties props); 例如：

```
Properties props=new Properties();
props.put("user", "wuwei");
props.put("password", "wuwei");
props.put("defaultRowPrefectch", "30");
props.put("dufaultBatchValue", "5");
Connection con=DriverManager.getConnection("jdbc.net/forum/images/smiles/
icon_surprised.gif border=0>racle:thin:@hostsString", props);
```

对象可以通过设置 setDefaultRowPrefetch(int) 和 setDefaultBatchValue(int) 两个参数类优化连接。

❑ 使用连接池

可以自己写一个连接池，这样程序的灵活性强，便于移植，设计了自己的连接池后， 在客户端调用建立对象。

（3）控制事务的提交

最好手动提交事务，不但可以保证数据原子性， 而且对性能提高留下余地。适当地选择事务的隔离级别：

❑ TRANSACTION_READ_UNCOMMITED（性能最高）。
❑ TRANSACTION_READ_COMMITED（快）。
❑ TRANSACTION_REFEATABLE_READ（中等）。
❑ RANSACTION_SERIALIZABLE（慢）。

（4）优化 ResultSet

批量读取数据。合理设置 ResultSet 的 getFetchSize()和 setFetchSize()方法中的参数， 使用正确的 get 和 set 方法，使用整数而不是字段名作为参数性能比较高，例如“setInt(1,100);”。

“setString(2,"aaaa");” 比 “setInt("id","100"); setString("name","aaaa");” 性能好。

（5）其他方面的性能优化

及时显示的关闭 Connection Statement ResultSet，其中 Connection 可以用 Connection Pool 处理。使用数据库系统的强大查询功能去组织数据，这样程序运行时和数据库服务的交互次数少，数据库返回给程序的记录条数少得多，所以性能有很大的提高。

6.7　小　　结

从本质上来说 JDBC 是一个基于 Java 的面向对象应用编程接口，描述了一套访问关系数据库的标准 Java 类库。JDBC 的总体结构由应用程序、驱动程序管理器、驱动程序和数据源 4 个组件构成。

ODBC 的基本思想是为用户提供简单、标准、透明的数据库连接的公共编程接口，为

应用程序提供了一套 CLI 函数库和基于 DLL 的运行支持环境。

数据库连接是一种关键的、有限的、昂贵的资源，数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，这项技术能明显提高对数据库操作的性能。

JDBC 程序的性能主要由两个因素决定：一是数据库本身的性质，另一个是与数据库相对独立的 JDBC 应用程序接口（API）的使用。

第 7 章　Connector/J 的使用

在本章中将结合大量程序代码，从实际应用的角度阐述 Connector/J 的相关知识。首先介绍 Connector/J 的安装，然后从 6 个方面展示如何进行 JDBC 编程，接着使用 Eclipse 工具结合运用 Struts、Hibernate 开源框架示例了一个符合 J2EE 规范的 Web 项目。

读者应扎实掌握 JDBC 的数据库编程基础知识，达到以不变应万变的功效。

7.1　安装 Connector/J

Connector/J 是专门针对 MySQL 而开发的 JDBC 驱动程序包。JDBC 是人们为了使用 Java 语言开发各种数据库应用软件而打包在一起的一些类。JDBC 与具体的数据通信库系统无关。因此，如果想通过 JDBC 去连接某个特定的数据库系统，就必须使用一个专为这种数据库系统而开发的 JDBC 驱动程序包，对 MySQL 数据库来说，这个驱动程序包就是 Connector/J。

7.1.1　支持的 Java 版本

MySQL Connector/J 支持 Java-2 JVMs，包括 JDK-1.2.x、JDK-1.3.x、JDK-1.4.x 和 JDK-1.5.x，并需要 JDK-1.4.x 或更新的版本进行编译。MySQL Connector/J 不支持 JDK-1.1.x 或 JDK-1.0.x。

由于实现了 java.sql.Savepoint，Connector/J 3.1.0 和更新版本不会运行在早于 1.4 版的 JDK 上，除非关闭了类验证器，这是因为类验证器将试图加载用于 java.sql.Savepoint 的类定义，除非使用了 savepoint 功能，否则驱动程序不会访问类验证器。

早于 1.4.x 版的 JVM 上，不能使用 Connector/J 3.1.0 或更高版本提供的新缓冲功能，这是因为该功能依赖在 JDK-1.4.0 中首次提供的 java.util.LinkedHashMap。

7.1.2　MySQL 服务器版本指南

MySQL Connector/J 支持所有著名的 MySQL 服务器版本。某些特性（外键，可更新结果集）需要更新的 MySQL 版本才能工作。

与 MySQL 服务器 4.1 版或更高版本建立连接时，最好使用 MySQL Connector/J 3.1 版以上的版本，这是因为它全面支持较新版本的服务器提供的特性，包括 Unicode 字符、视图、存储程序和服务器端预处理语句。尽管 3.0 版 Connector/J 能够与 MySQL 服务器 4.1 或更高

版本建立连接，但由于实现了 Unicode 字符和新的鉴定机制，将无法更新 Connector/J 3.0 以支持当前和未来服务器版本中提供的新特性。

7.1.3　Connector/J 的安装

Connector/J 软件包可以从 http://dev.mysql.com/downloads/connector/j/5.0.html 下载到。可以看到网站上提供了若干版本的 Connector/J 软件包，其中 5.1 版本是 Beta 版，因此在本书中选用的是较为成熟的 5.0.7 版本。针对不同操作系统，分别提供下载链接，下载到的有*.ZIP（适用于 Windows 系统）和*.tar.gz（适用于 UNIX/Linux 系统）两种格式的压缩文档。这两个压缩文档的内容其实是一些同样的文件（Java 是与平台无关的）。在安装 Connector/J 的第一步，必须先把下载下来的压缩文档解压到一个选定的子目录中。例如，现在是在一台 Windows XP 的系统上安装，将下载下来的软件包 mysql-connector-java-5.0.7.zip 解压到指定的盘符下，文件目录结构如图 7-1 所示。Doc 目录下存放的是英文版的 Connector/J 的相应文档，有 PDF 和 HTML 两种格式。Src 目录下存放的是类的 Java 原码，它的目录结构也就是包的层次结构，如果要做深入的研究，是非常有用的。在使用中真正起作用的是 mysql-connector-java-5.0.7-bin.jar，但是如何才能让 Java 引擎能够在执行 Java 程序时找到新安装的驱动程序库。最简单的方法是把 mysql-connector-java-5.0.7-bin.jar 文件复制到 Java 安装目录的 jre\lib\ext 中去（如图 7-1 所示），Java 程序在执行时会自动到这个地方来寻找驱动程序。

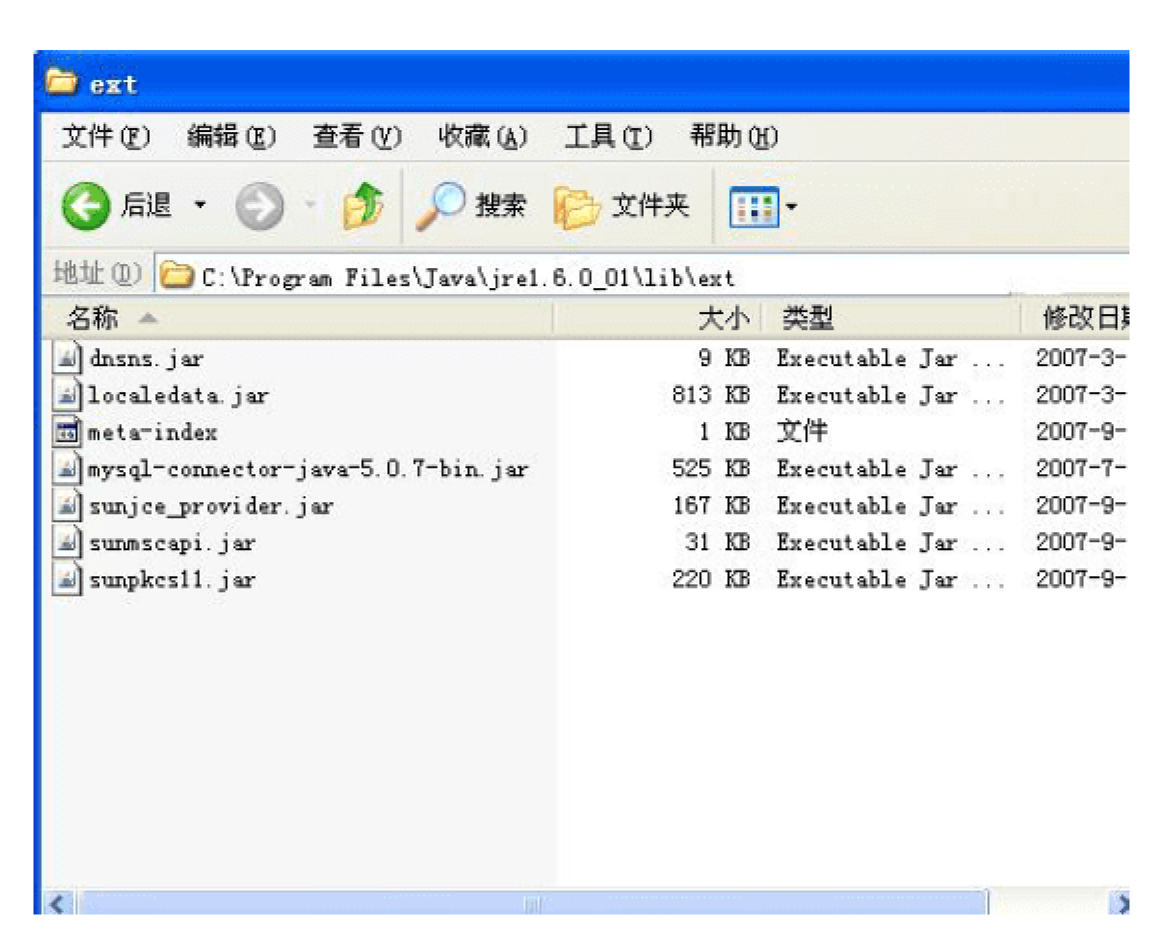


图 7-1　Connector/J 安装目录

接下来就要修改 CLASSPATH 环境变量。Java 程序在执行时会到这个环境变量所列出的各个目录中寻找驱动程序。也就是说，如果想让 Java 程序使用 Connector/J，就必须把 mysql-connector-java-5.0.7-bin.jar 文件所在的目录添加到这个环境变量中。这里有一点需要大家特别注意：如果想让自己能够执行当前目录中的 Java 程序，就必须让这个环境变量包含路径“.”。

在 Windows 系统中，可以用 DOS 命令临时改变 CLASSPATH 变量的设置值。如果想永久地改变它，在 Windows 2003/XP 系统中，可以能过菜单命令 Control Panel（控制面板）

→System（系统）→Advanced（高级）→Environment Variables（环境变量）打开的对话框进行。CLASSPATH 变量中路径必须用分号（“;”）隔开。也可以用下面的这条命令追加 CLASSPATH 变量的配置：

```
Set classpath=%classpath%; C:\Program Files\Java\jre1.6.0_01\lib\ext\
mysql-connector-java-5.0.7-bin.jar
```

在 Linux 系统中，可以用 Export classpath=xx 临时改变 CLASSPATH 变量的设置值，对它进行永久性修改必须通过 profile 文件来进行（Linux 版本不同，文件所在位置也不同）。也可以在启动 JVM（Java 虚拟机）时用命令行开关“-cp”直接指定它，通过该方式安装驱动。

如果只是出于测试目的，还可以采用这样一种临时办法：把 Connector/J 软件包中的 com 和 org 目录复制到相应子目录（也就是开发的 Java 程序所在的那个目录），最后可用下面这个小程序来测试 Connector/J 是不是安装成功。它应该不返回任何出错消息，只在屏幕上显示 ok。

例 7-1　测试 Connector/J 安装结果。

```
/* example file DatabaseTest.java */
import java.sql.*;
public class DatabaseTest
{
   public static void main(String[] args)
  {
    try
    {
           Drive d =(Driver)
           Class.forName("com.mysql.jdbc.Driver").newInstance();
           System.out.println("ok");}
      catch(Exception e){
           System.out.println("Error:  ")+e.toString());
     }
   }
}
```

Connector/J 安装失败的最常见的原因是 Java 引擎找不到 Connector/J 驱动程序的缘故，典型的错误提示是：java.lang.ClassNotFoundException:com.mysql.jdbc.Driver（未能找到 JDBC 驱动程序）。

7.2　JDBC 引用

在第 6 章介绍了 JDBC 的基本知识，包括 JDBC 的来源、作用、体系结构以及它的 API，在本章将以 MySQL 数据库为例，介绍利用 JDBC 进行数据库编程的相关知识。JDBC 的应用可以形象地用图 7-2 来表达。

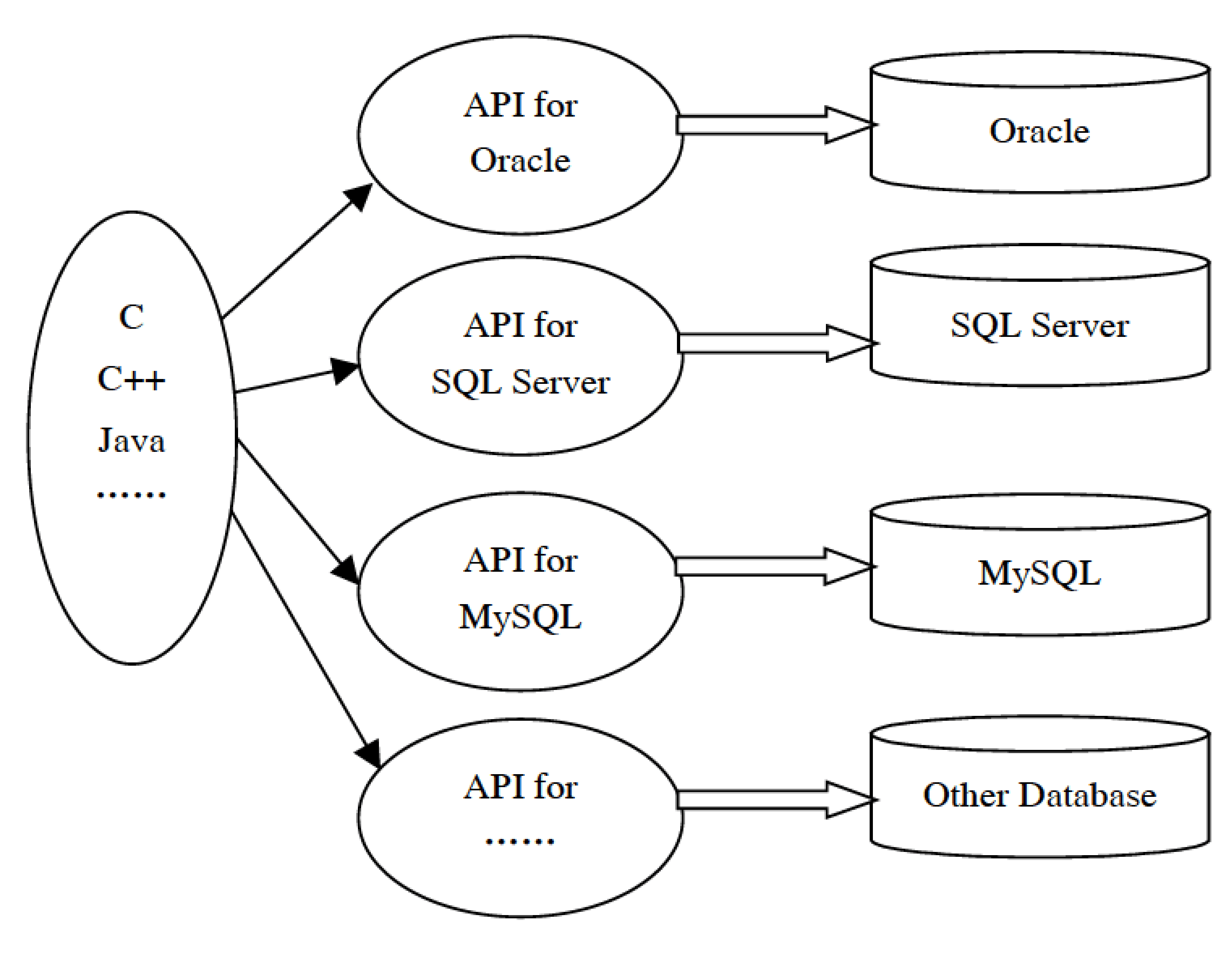


图 7-2　JDBC 接口

JDBC 在 Java 应用程序和数据库这两端分别提供了接口，如果连接的是 Oracle 数据库，就用 Oracle 的 JDBC 类库；如果连接的是 MySQL 数据库，就用 MySQL 的 JDBC 类库；如果连接的是 SQL Server 数据库，就用 SQL Server 的 JDBC 类库。但是这些类库有个共同的特点，给 Java 程序员提供了统一的编程接口，而屏蔽掉了针对不同数据库的 JDBC 类库的差异。

举个形象的例子来说明这个问题：将不同厂家生产的 mp3 比作数据库，mp3 数据线比作 JDBC，mp3 数据线有两端，与 mp3 相连接的那端各不相同，因 mp3 的品牌而异，这就像 JDBC 与数据库相连的那端，JDBC 的类库因数据库产品而异。但与 USB 接口相连接的那一端是相同的，符合 USB 接口标准，这就像 JDBC 与 Java 程序相连的那端，给 Java 程序员提供了统一的接口，就像计算机生产厂家在设计 USB 接口时，不需要去考虑各种外设设备的接口而只要符合 USB 标准即可。同样的道理，在开发 Java 程序时，不管访问何种数据库，只要遵照 JDBC API 提供的接口即可。

7.2.1　JDBC 基本编程的步骤

JDBC 编程难吗？不难，它有固定的步骤，在各个应用程序中万变不离其宗，只要扎扎实实地掌握了各个步骤，今后就能应对各种各样的 JDBC 编程，可谓是“一次掌握，到处运用”。

JDBC 编程的步骤可以分为以下 6 步：

（1）Load the Driver（加载驱动）

其使用的方法有：

Class.forName() 或者

Class.forName().newInstance() 或者

New DriverName()

（2）Connect to the DataBase（连接数据库）

DriverManager.getConnection()

（3）Execute the SQL（执行 SQL 语句）

Connection.CreateStatement()

Statement.executeQuery()

Statement.executeUpdate()

（4）Retrieve the result data（取得结果集）

循环取得 while(rs.next())

（5）Show the result data（显示数据）

将数据库中的各种类型转化为 java 中的类型（getXXX）方法

（6）Close（关闭）

close the resultset.

Close the statement

Close the connection

【特别提示】关于步骤（1），我们知道，Java 程序要连接数据库，必须先找到相应数据库的 JDBC 类库，才能进一步找到相应的驱动，这是一个名为 Driver 的类，通过它，Java 应用程序就能连接到自己的数据库上。对于 Java 系统来讲，JDBC 类库有很多种，不同的数据库有不同的 JDBC 类库，那么 Java 程序如何知道是哪种呢？DriverManager 相当于一个管理数据库的大管家，通过向 DriverManager 注册相应数据库的驱动程序类来实现。

下面以 MySQL 数据库为例，开始 JDBC 编程：

启动 MySQL 数据库的“MySQL Command Line Client”。

在 Enter password:后输入密码，成功登录后出现欢迎信息，在 mysql>后输入“use mysql”，显示“Database changed”，则表明可以使用 MySQL 数据库了，在 mysql>后输入创建表的语句：

```
create table DBTest(
id int primary key,
name char(8),
age int
);
```

显示“Query OK,0 rows affected (0.13 sec)”，说明创建表成功，用时 0.13 秒。接下来插入一条记录，在 mysql>后输入“insert into DBTest values(1,‘wpm’,25);”，显示“Query ok,1 rows affected(0.06 sec)”，说明插入成功，再插入一条记录“insert into DBTest values (2,‘nlh’,28);”，输入“select * from DBTest;”即可看到刚插入的记录，如图 7-3 所示。

用编辑工具建立的 Java 文件 DatabaseTest.java。

```
mysql> create table DBTest(
    -> id int primary key,
    -> name char(8),age int);
Query OK, 0 rows affected (0.13 sec)

mysql> insert into dbtest values(1,'wpm',25);
Query OK, 1 row affected (0.06 sec)

mysql> insert into dbtest values(2,'nlh',28);
Query OK, 1 row affected (0.02 sec)

mysql> select * from dbtest
    -> ;
+----+------+------+
| id | name | age  |
+----+------+------+
|  1 | wpm  |   25 |
|  2 | nlh  |   28 |
+----+------+------+
2 rows in set (0.02 sec)

mysql>
```

图 7-3　插入记录后查询

例 7-2　测试数据库连接。

```
import java.sql.*;
public class DatabaseTest
{
  public static void main(String[] args) throws Exception
  {

        Class.forName("com.mysql.jdbc.Driver");
        String url="jdbc:mysql://127.0.0.1:3306/mysql";
        String  user="root";
        String password="admin";
        Connection conn=DriverManager.getConnection(url, user, password);
        Statement stmt=conn.createStatement();
        String query="select * from dbtest";
        ResultSet rs=stmt.executeQuery(query);
        while(rs.next())
        {
          System.out.println(rs.getString(1));
          System.out.println(rs.getString(2));
          System.out.println(rs.getString(3));
          System.out.println(" ");
        }
        rs.close();
        stmt.close();
        conn.close();
  }
}
```

保存在 E:\下，编译运行程序，执行成功，如图 7-4 所示。

下面是对程序的解释：

（1）Class.forName 中的 Class 是 java.lang 包下的一个专门的类，Class.forName("com.mysql.jdbc.Driver")的作用可以理解为根据该字符串在内存中创建这个字符串标识的

类的实例，即在内存中分配了一个对象，这个对象就是 com.mysql.jdbc.Driver 的实例，如果找不到字符串标识的类，就会抛出异常 ClassNotFoundException，需要注意的是，Class.forName("com.mysql.jdbc.Driver") 创建的 com.mysql.jdbc.Driver 实例自动向 DriverManager 注册了。也可以用 new com.mysql.jdbc.Driver(); 或者 Class.forName ("com.mysql.jdbc.Driver").newInstance();来代替 Class.forName ("com.mysql. jdbc.Driver");，它们的作用完全一样。

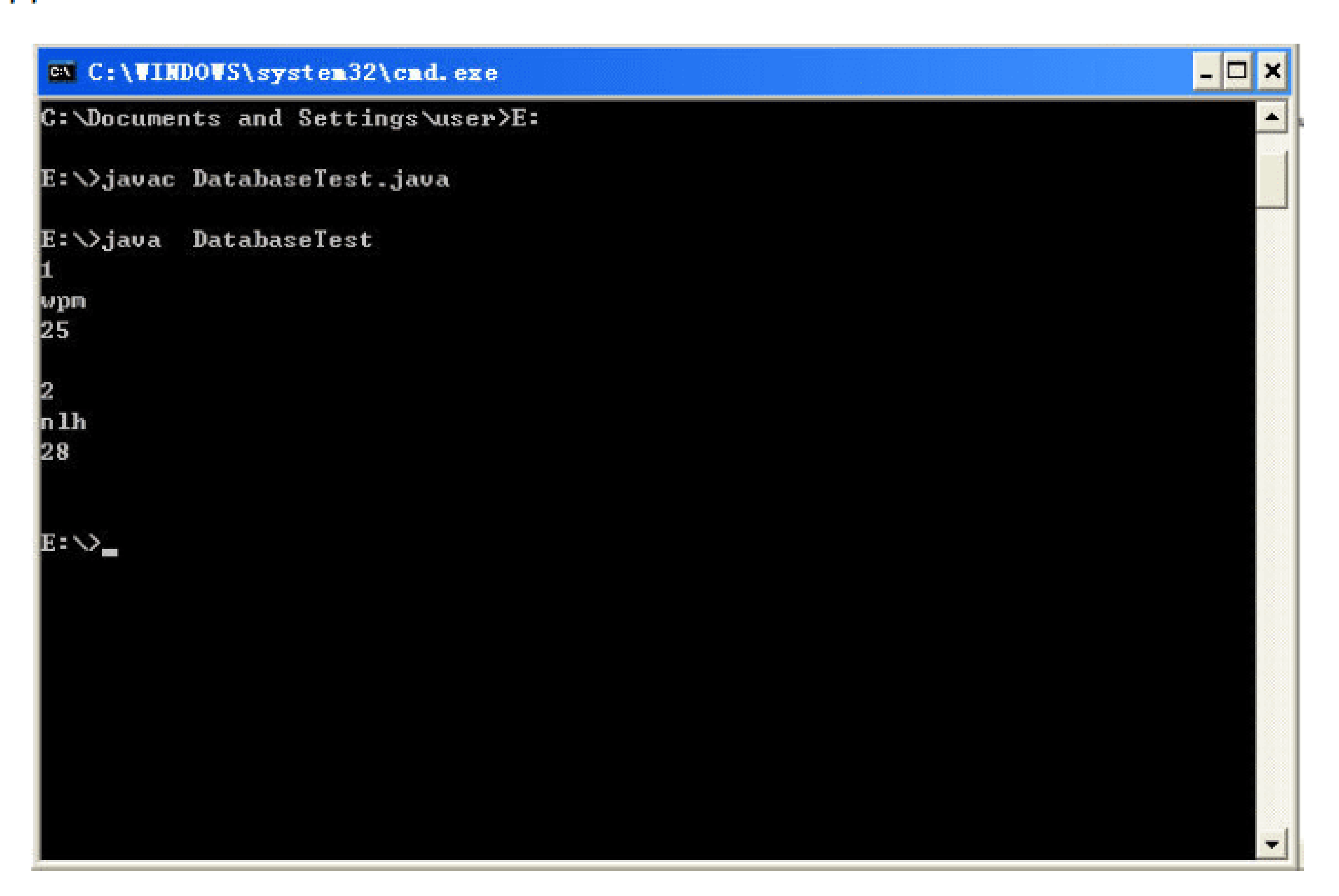


图 7-4　程序执行结果

（2）JDBC URL 用于标识一个被注册的驱动程序，驱动程序管理器 DriverManager 通过这个 URL 选择正确的驱动程序，从而建立到数据库的连接，JDBC URL 的语法如下：

jdbc:subprotocol:subname，整个 URL 用冒号（:）分为了 3 个部分。

- 协议：在上面的语法中，jdbc 为协议，在 JDBC 中，它是唯一允许的协议。
- 子协议：用于标识一个数据库驱动程序。
- 子名称：子名称的语法与具体的驱动程序相关，驱动程序可以选择任何形式的适合其实现的语法。

一些常用的数据库的 JDBC URL 形式如下。

- MySQL：jdbc:mysql://localhost:3306/databasename。
- Oracle：jdbc:oracle:thin:@localhost:1521:ORCL。
- SQL Server 2000：Jdbc:microsoft:sqlserver://localhost:1433;databasename=pubs。

另外，也可以通过 JDBC-ODBC 桥的方式访问数据库，这种形式加载驱动程序的类是 sun.jdbc.odbc.JdbcOdbcDriver。

JDBC URL 是 jdbc:odbc:datasource_name。

【特别提示】3306、1521、1433 指端口号，各类型数据库各不相同；pubs 用实际的数据库名代替，ORCL 用实例名代替。

stmt.executeQuery(query)执行后返回一个结果集，放在了 ResultSet 对象中，rs 就像一个游标，它与数据库游标的一个区别是：数据库的游标指向第一条记录，而这里的 rs 指在

结果集第一条记录的前面一个位置上，while 语句的作用是循环取得结果集的记录，rs.next()方法表示，游标向后移动一个位置，如果该位置有记录，就返回 true，否则返回 false。rs.getXXX()方法是取当前记录的字段，参数可以是字段名，也可以是字段序号，第一个字段为 1，第二个字段为 2，依此类推，rs.getString(1)表示把当前记录的第一个字段的值取出并转化为 String 类型，类似的还有 getInt()、getBlob()方法等，具体的内容可以查阅 Java API，原则是字段的值的类型必须可以转化为 getXXX()方法的类型。

关闭连接，后打开的先关闭。

以上是一个典型的 JDBC 编程的例子，这个程序有什么问题吗？

假设在 while 循环中出现了异常怎么办？那么后面的关闭语句就执行不到了，分配的内存就释放不了，造成资源的浪费，如果有很多个客户连接到服务器，那么这就有可能给应用程序带来灾难性的后果，一定要养成良好的编程习惯。DatabaseTest2.java 文件可以改造为如下所示。

例 7-3　规范的数据库连接。

```
import java.sql.*;

public class DatabaseTest2
{
    public static void main(String[] args)
    {
         ResultSet rs=null;
         Statement stmt=null;
         Connection conn=null;
         try
         {

          Class.forName("com.mysql.jdbc.Driver");
          String url="jdbc:mysql://127.0.0.1:3306/mysql";
          String  user="root";
          String password="admin";
          conn=DriverManager.getConnection(url, user, password);
          stmt=conn.createStatement();
          String query="select * from dbtest";
          rs=stmt.executeQuery(query);
          while(rs.next())
          {
              System.out.println(rs.getString(1));
              System.out.println(rs.getString(2));
              System.out.println(rs.getString(3));
              System.out.println(" ");
          }
         }
         catch(ClassNotFoundException e)
         {
            e.printStackTrace();
```

```
            }
            catch(SQLException e)
            {
              e.printStackTrace();
            }
            finally
            {
                 try
                 {
                   if(rs!=null)
                   {
                          rs.close();
                   }
                   if(stmt!=null)
                   {
                         stmt.close();
                   }
                   if(conn!=null)
                   {
                         conn.close();
                   }
                 }
                 catch(SQLException e)
                 {
                        e.printStackTrace();
             }
          }
      }
}
```

仍然需要说明一点，在实际开发中，e.printStackTrace()不应该出现在程序中，一方面暴露程序的错误是危险的事情；另一方面起不到记录错误的效果，不利于修改。通常的做法是把异常写进错误日志。

下面介绍数据库的 DML 操作。假设需要接受用户的输入，然后插入到 DBTest 表中，可以按如下所示实现程序，建立文件 DMLTest.java。

例 7-4　数据库的 DML 操作。

```
import java.sql.*;
public class DMLTest
{
    public static void main(String[] args)
    {
        if(args.length!=3)
        {
            System.out.println("参数个数错误!，请重新输入!");
            System.exit(-1);
        }
        int id=0;
```

```
        try
        {
          id=Integer.parseInt(args[0]);
        }
        catch(NumberFormatException e)
        {
            System.out.print("第一个参数请输入整数");
            System.exit(-1);
        }
        String name=args[1];
        int age=0;
        try
        {
          age=Integer.parseInt(args[2]);
        }
        catch(NumberFormatException e)
        {
            System.out.print("第 3 个参数请输入整数");
            System.exit(-1);
        }
        Statement stmt=null;
        Connection conn=null;
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            String url="jdbc:mysql://127.0.0.1:3306/mysql";
            String  user="root";
            String password="admin";
            conn=DriverManager.getConnection(url, user, password);
            stmt=conn.createStatement();
            String sql="insert into dbtest values("+id+",'"+name+"',
            "+age+")";
            stmt.executeUpdate(sql);

        }
         catch(ClassNotFoundException e)
         {
             e.printStackTrace();
         }
         catch(SQLException e)
         {
             e.printStackTrace();
         }
         finally
         {
             try
             {
                 if(stmt!=null)
                 {
```

```
                stmt.close();
            }
            if(conn!=null)
            {
                conn.close();
            }
        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
    }
  }
}
```

需要解释一下：

（1）用 Java 命令运行带输入参数的程序时，格式如下：

java　DMLTest　5　nlh　25

（2）对于初学者，一定要注意 String sql="insert into dbtest values("+id+",'"+name+"',"+age+")";这个语句的含义，切不能写成 String sql="insert into dbtest values(id,'name','age')"。

（3）调试 SQL 语句时，建议先用 System.out.println(sql);打印出 SQL 语句，直接在数据库中执行。

（4）局部变量的定义位置，以前一直提倡在方法的开始位置定义变量，但是现在也有不少用户建议在其他位置定义。

在这种情况下，SQL 语句书写很麻烦，还容易出错，效率也不是很高，下面解释另一种 Statement。

7.2.2　预处理语句

从 JDBC API 可以看到，PreparedStatement 扩展自 Statement，PreparedStatement 是用于执行预编译的 SQL 语句，在提高性能方面有很多优点。下面以 PreparedStatement 举个例子，保存为 TestPrepStmt.java。

例 7-5　PreparedStatement 预处理语句。

```
import java.sql.*;
public class TestPrepStmt
{
    public static void main(String[] args)
    {
        if(args.length!=3)
        {
            System.out.println("参数错误!,请重新输入!");
            System.exit(-1);
```

```
        }
        int id=0;
        try
        {
          id=Integer.parseInt(args[0]);
        }
        catch(NumberFormatException e)
        {
            System.out.print("请输入整数");
            System.exit(-1);
        }
        String name=args[1];
        int age=0;
        try
        {
          age=Integer.parseInt(args[2]);
        }
        catch(NumberFormatException e)
        {
            System.out.print("请输入整数");
            System.exit(-1);
        }
        PreparedStatement pstmt=null;
        Connection conn=null;
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            String url="jdbc:mysql://127.0.0.1:3306/mysql";
            String  user="root";
            String password="admin";
            conn=DriverManager.getConnection(url, user, password);
            pstmt=conn.prepareStatement("insert into dbtest values(?,?,?)");
            pstmt.setInt(1, id);
            pstmt.setString(2,name);
            pstmt.setInt(3,age);
            pstmt.executeUpdate();

        }
         catch(ClassNotFoundException e)
         {
             e.printStackTrace();
         }
         catch(SQLException e)
         {
             e.printStackTrace();
         }
         finally
         {
             try
             {
```

```
                if(pstmt!=null)
                {
                    pstmt.close();
                }
                if(conn!=null)
                {
                    conn.close();
                }
            }
            catch(SQLException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

这个程序在运行时，输入参数示例如下：

java　TestPrepStmt　6　 wpm　25

7.2.3　批处理命令

执行批处理很简单，举例子来说明使用方法。

例 7-6　statement 语句的批处理。

```
import java.sql.*;

public class TestBatchforStatement
{
    public static void main(String[] args)
    {

        Statement stmt=null;
        Connection conn=null;
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            String url="jdbc:mysql://127.0.0.1:3306/mysql";
            String  user="root";
            String password="admin";
            conn=DriverManager.getConnection(url,user,password);
            stmt=conn.createStatement();
            stmt.addBatch("insert into dbtest values(8,'nlh',25)");
            stmt.addBatch("insert into dbtest values(9,'wpm',25)");
            stmt.addBatch("insert into dbtest values(10,'nba',25)");
            stmt.executeBatch();

        }
```

```
            catch(ClassNotFoundException e)
            {
                e.printStackTrace();
            }
            catch(SQLException e)
            {
                e.printStackTrace();
            }
            finally
            {
                try
                {
                    if(stmt!=null)
                    {
                        stmt.close();
                    }
                    if(conn!=null)
                    {
                        conn.close();
                    }
                }
             }
             catch(SQLException e)
             {
                    e.printStackTrace();
             }
        }

    }
}
```

举例说明 PreparedStatement 语句的批处理：建立文件 TestBatchforPstmt.java，内容如下所示。

例 7-7　PreparedStatement 语句的批处理。

```
import java.sql.*;

public class TestBatchforPstmt
{
    public static void main(String[] args)
    {

        PreparedStatement pstmt=null;
        Connection conn=null;
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            String url="jdbc:mysql://127.0.0.1:3306/mysql";
            String  user="root";
```

```
        String password="admin";
        conn=DriverManager.getConnection(url,user,password);
        pstmt=conn.prepareStatement("insert into dbtest values(?,?,?)");
        pstmt.setInt(1,15);
        pstmt.setString(2,"nlh");
        pstmt.setInt(3,25);

        pstmt.setInt(1,16);
        pstmt.setString(2,"nlh");
        pstmt.setInt(3,25);

        pstmt.setInt(1,17);
        pstmt.setString(2,"nlh");
        pstmt.setInt(3,25);


        pstmt.executeBatch();

    }
        catch(ClassNotFoundException e)
        {
            e.printStackTrace();
        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                if(pstmt!=null)
                {
                    pstmt.close();
                }
                if(conn!=null)
                {
                    conn.close();
                }
            }
            catch(SQLException e)
            {
                e.printStackTrace();
            }
        }

    }

}
```

7.2.4　事务

举例说明 JDBC 处理 Transaction：建立文件 TestTransaction.java，内容如下所示。

例 7-8　JDBC 处理 Transaction。

```
import java.sql.*;
public class TestTransaction {


    public static void main(String[] args) {

        Connection conn = null;
        Statement stmt = null;

        try {
            Class.forName("com.mysql.jdbc.Driver");
            String url="jdbc:mysql://127.0.0.1:3306/mysql";
            String  user="root";
            String password="admin";
            conn=DriverManager.getConnection(url, user, password);
            conn.setAutoCommit(false);
            stmt = conn.createStatement();
            stmt.addBatch("insert into dbtest values (151,'nlh',25)");
            stmt.addBatch("insert into dbtest values (152,'nlh',25)");
            stmt.addBatch("insert into dbtest values (153, 'nlh',25)");
            stmt.executeBatch();
            conn.commit();
            conn.setAutoCommit(true);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch(SQLException e) {

            e.printStackTrace();

            try {
                    if(conn != null)
                    {
                        conn.rollback();
                        conn.setAutoCommit(true);
                    }
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }finally {
            try {
                    if(stmt != null)
                        stmt.close();
```

```
                    if(conn != null)
                        conn.close();
            } catch (SQLException e) {
                    e.printStackTrace();
            }
        }
    }

}
```

举例说明 JDBC 处理可滚动的结果集：建立文件 TestScroll.java，内容如下所示。

例 7-9　JDBC 处理可滚动的结果集。

```
import java.sql.*;

public class TestScroll
{
    public static void main(String args[])
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            String url="jdbc:mysql://127.0.0.1:3306/mysql";
            String  user="root";
            String password="admin";
            Connection conn=DriverManager.getConnection(url, user, password);
            Statement stmt = conn.createStatement
            (ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
            ResultSet rs = stmt.executeQuery("select * from dbtest order by
            id");
            rs.next();
            System.out.println(rs.getInt(1));
            rs.last();
            System.out.println(rs.getString(1));
            System.out.println(rs.isLast());
            System.out.println(rs.isAfterLast());
            System.out.println(rs.getRow());
            rs.previous();
            System.out.println(rs.getString(1));
            rs.absolute(6);
            System.out.println(rs.getString(1));
            rs.close();
            stmt.close();
            conn.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
```

```
    }
}
```

7.2.5　可更新的结果集

举例说明 JDBC 处理可更新的结果集：建立文件 TestUpdataRs.java，内容如下所示。

例 7-10　JDBC 处理可更新的结果集。

```
import java.sql.*;
public class TestUpdataRs
{
    public static void main(String args[])
    {
        try
        {
          Class.forName("com.mysql.jdbc.Driver");
          String url="jdbc:mysql://127.0.0.1:3306/mysql";
          String  user="root";
          String password="admin";
          Connection conn=DriverManager.getConnection(url,user,password);
          Statement stmt=conn.createStatement(ResultSet.TYPE_SCROLL_
          INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
          ResultSet rs=stmt.executeQuery("select * from dbtest");
          rs.next();
         //更新一行数据
         rs.updateString("name","AAAA");
         rs.updateRow();

         //插入新行
         rs.moveToInsertRow();
         rs.updateInt(1,9999);
         rs.updateString("name","nlh");
         rs.updateInt(3, 25);

         rs.insertRow();
         //将光标移动到新建的行
         rs.moveToCurrentRow();

         //删除行
         rs.absolute(5);
         rs.deleteRow();

         //取消更新
         //rs.cancelRowUpdates();

         }
         catch(Exception e)
         {
```

```
                e.printStackTrace();
            }
        }
    }
```

7.2.6　用 DataSource 建立连接

随着 J2EE 标准的出现，JDBC 增加了两种新的接口：DataSource 和 RowSet，DataSource 的属性可以动态改变。RowSet 扩展了 ResultSet，支持断开的结果集，支持 javaBean 标准。在 javax.sql.*包中， DataSource 是 DriverManager 的替代，可以实现更多高级的功能：连接池、分布式处理（用同一个数据库操作对来自多个数据库的数据进行处理）等。

下面的示例只是简单地演示了如何使用 DataSource.getConnection()方法创建与数据库连接，不涉及那些高级的功能。首先，创建一个 com.mysql.jdbc.jdbc2.optional. Mysql DataSource 类的对象，然后用 setServerName()和 setDatabaseName()方法分别设置一个主机名和一个数据库名，最后用 getConnection()方法去建立与数据库的连接。在调用 getConnection()方法时只需给出用户名和密码即可。这个方法将返回一个 Connection 对象，后续的数据库操作都需要通过这个 Connection 对象来进行。下面是部分程序。

例 7-11　用 DataSource 建立连接。

```
    import java.sql.*;
    import javax.sql;
    public class DsConnection
    {
       public static void main(String[] args)
       {
      try{
             com.mysql.jdbc.jdbc2.optional.MysqlDataSource ds;
        Connection conn2;
        ds=new com.mysql.jdbc.jdbc2.optional.MysqlDataSource();
        ds.setServerName("myhost");
           //设置主机名
           ds.setDatabaseName("mysql");
           //设置数据库名
        conn2=ds.getConnection("root","admin");

          //下略,与 DriverManager 方式相同
        }
     catch(Exception e){
       system.out.println("Error:"+e.toString());
      }
     }
    }
```

7.3　与 J2EE 和其他 Java 框架一起使用 Connector/J

JDBC 可以做如下 3 件事情：

（1）与数据库建立连接。

（2）发送 SQL 语句。

（3）处理结果。

JDBC 是个“低级”接口，它用于直接调用 SQL 命令。在这方面它的功能极佳，并比其他的数据库连接 API 易于使用，但它同时也被设计为一种基础接口，在它之上可以建立高级接口和工具。实际上，使用 JDBC 对数据库中的数据进行增加、删除和修改的操作就是持久化的过程。

然而，直接使用 JDBC 作为持久层有一些难以解决的问题，这些问题包括繁琐的代码、表间连接、级联以及层与层之间的耦合严重等问题，在这种情况下就需要 ORM 来对数据库进行持久化。

7.3.1　O/R Mapping 的介绍

O/R Mapping（Object-Relational Mapping，ORM），其作用是在关系型数据库和类对象之间做映射。通过这个映射，做具体数据库操作时，就不用再和复杂的 SQL 语句打交道，只要像平时操作对象一样来操作即可。ORM 通过将面向对象模型映射到关系模型，使得程序能直接操作对象模型。利用 ORM 能够将实体对象数据自动存入关系数据库中。ORM 除了提高重用性以外，ORM 还有以下好处：

（1）提高学习和开发效率，降低开发成本。

（2）简化代码，减少 Bug 数量。

（3）提高性能。

通过 Cache，能够对性能进行调优。ORM 隔开了数据存储层和业务逻辑层，这样就能对每一层进行单独跟踪，以找出效率瓶颈，优化性能。

（4）方便数据库转换。

ORM 将业务层与数据存储隔开，对于业务逻辑来说，具体的数据对其是不可见的，所以开发人员不需要关心底层的数据存储。例如，把 SQL Server 数据转换成 Oracle 数据库时，只需要修改配置文件即可。

现有的 ORM 框架很多，除了应用最广泛的 Hibernate，还有 LBLGen Pro、Gentle.NET、Entity Broker 和 XPO.NET。

7.3.2　Hibernate 介绍

Hibernate 是个开放源代码的 O/R Mapping 框架，它对 JDBC 进行了非常轻量级的对象

封装，使 Java 程序员可以方便地使用对象编程思想来操纵数据库。Hibernate 可以应用在任何使用 JDBC 的场合，既可以在 Java 的客户端程序使用，也可以在 Serverlet/JSP 的 Web 应用程序中使用。

现在 Hibernate 可以和多种 Web 服务器或者应用服务器良好集成，支持几乎所有流行的数据库服务器。最具革命意义的是，Hibernate 可以在应用 EJB 的 J2EE 架构中取代 CMP，担负数据持久化的重任。图 7-5 显示了 Hibernate 在系统中所处的位置。

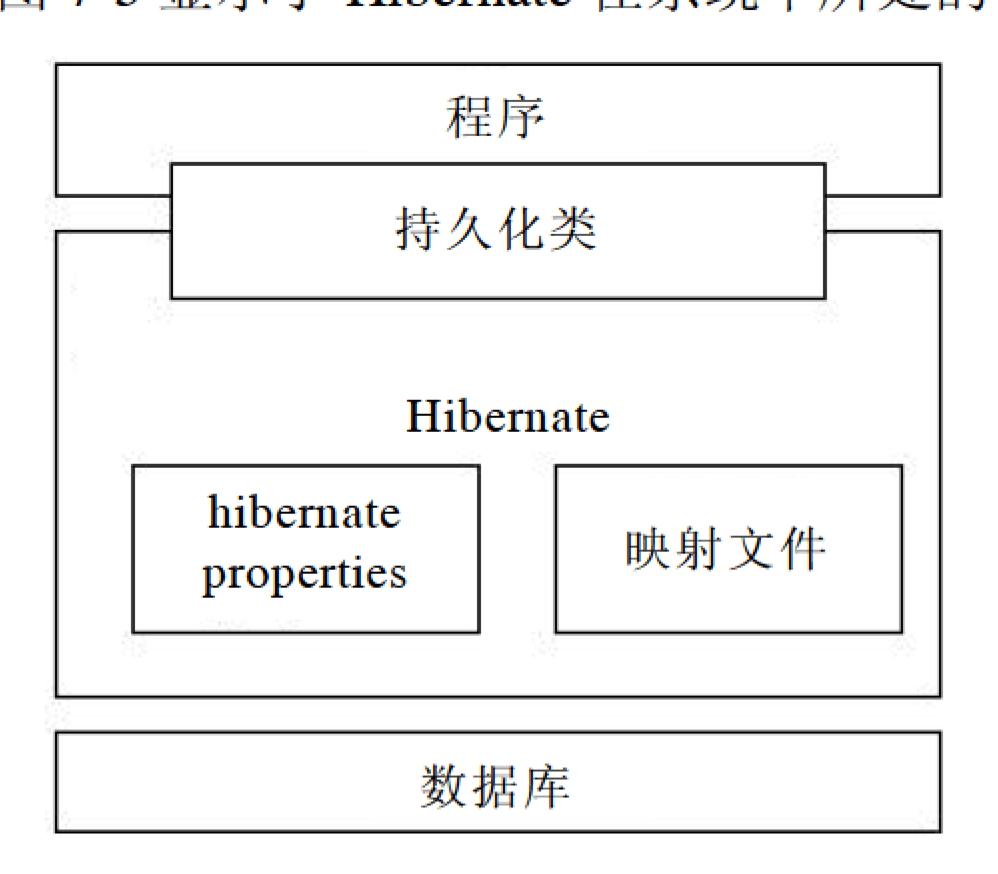


图 7-5　Hibernate 在系统中的位置

从图中可以看到，Hibernate 处于数据库和应用程序之间，为应用程序提供了可操作的持久化对象。从上层的角度看，数据库是透明地利用映射操作数据，使用 Hibernate 配置文件配置 session-config 等信息。图中的配置方式是 hibernate.properties，但实际上，更多采用的是 hibernate.cfg.xml。

7.3.3　Struts 简介

Struts 用<data-sources>元素可以定义多个数据源。由于使用了 Hibernate 做数据操作，所以在本书实例中，不需要设定<data-sources>来定义数据源。但是为了讲解的完整性，并考虑到某些情况下的需要，在这里作简单介绍。

配置 JDBC 数据源

下面是一个用<data-sources>定义数据源的例子。<data-sources>中可以定义多个<data-sources>元素来指定数据源，每个<data-sources>元素通过指定数据库 JDBC 类、数据库地址和用户名密码等属性来定义数据源。在这个例子中，定义了一个 id 为 DS1，key 为 conPool（这个 key 是用来给 Action 使用的，Action 类使用这个名称来寻找连接）、类型为 org.apache.struts.util.GenericDataSource、JDBC 类为 org.test.mm.mysql.Driver、数据库地址为 jdbc:mysql://localhost/test、用户名为 abc、密码为 123 的数据源，同时设定了这个数据源的自动数据库更新模式、最大连接数、最小连接数等属性。

```
<data-sources>
    <!--  可定义多个 data-source，并用 key、type 等字段定义连接属性  -->
  <data-soruce id="ds1"key="conPool"type="org.apache.struts.util.
```

```
GenericDataSoruce"autoCommit="true"description="Description"driverClass
="org.test.mm.mysql.Driver" maxCount="4" minCount="2" url="jdbc:mysql:
//localhost/test" user="abc" password="123">
<data-source/>
<data-sources>
```

表 7-1 描述了<data-sources>元素的各个属性及其含义。

表 7-1　<data-sources>元素的各个属性及其含义

属　性	描　述
id	标识
key	Action 类使用这个名称来寻找连接
type	实现 JDBC 接口的类的名称
desciption	数据源的描述
autoCommit	数据源创建的连接所使用的自动更新数据库模式
driverClass	数据源所使用的类，用来显示 JDBC 驱动程序接口
loginTimeout	数据库时间的限制，以秒为单位
maxCount	最大连接数
minCount	最小连接数
password	数据库访问密码
readOnly	是否只读的连接
user	访问数据库的用户名
url	JDBC 的 URL

配置完成以后，通过指定的 key，Action 类即可访问数据源了，例如以下代码：

```
Javax.sql.DataSource ds = servlet.findDataSource("conPool");//找到此 data-
source 配置信息
Javax.sql.connection con = ds.getConnection();//连接
```

下面以 Struts 和 Hibernate 结合为例，介绍如何开发 Web 应用程序。现在开始具体的实施过程。

（1）启动 Eclipse，新建一个工程。

（2）导入一些包和 tld 文件到工程中，并修改 web.xml 文件，以及编写 struts-config.xml 配置文件，以支持 Struts。

（3）修改完成后的 web.xml 文件内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app id="WebApp">
   <display-name>Example1</display-name>


   <servlet>
      <servlet-name>action</servlet-name>
```

```
        <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
        <init-param>
            <param-name>config</param-name>
            <param-value>/WEB-INF/struts-config.xml</param-value>
        </init-param>
        <init-param>
            <param-name>debug</param-name>
            <param-value>2</param-value>
        </init-param>
        <init-param>
            <param-name>detail</param-name>
            <param-value>2</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
    </servlet>

     <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>login.jsp</welcome-file>
    </welcome-file-list>

    <!-- Standard Action Servlet Configuration (with debugging) -->


    <!-- Standard Action Servlet Mapping -->

    <taglib>
        <taglib-uri>/tags/struts-bean</taglib-uri>
        <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
    </taglib>

    <taglib>
        <taglib-uri>/tags/struts-html</taglib-uri>
        <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
    </taglib>

    <taglib>
        <taglib-uri>/tags/struts-logic</taglib-uri>
        <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
    </taglib>

    <taglib>
        <taglib-uri>/tags/struts-nested</taglib-uri>
        <taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
    </taglib>

</web-app>
```

下面简单介绍上述代码的含义：

① Display-name 代表这个 Web 应用的名称，用户在访问特定的 URL 时需要在前面指明这个名字，例如 http://localhost:8080/Example1/a.jsp；另一种办法是在服务器的配置选项中通过这个名字，将某一 Web 应用设置为默认。

② 在 Servlet 标签中声明了对 Struts 的支持。

③ 在 Servlet-Mapping 中指定访问 Struts 的 URL:的后缀，用 do 来访问。

④ 声明 4 个 Struts 标签库以及 tld 的位置。

（4）建立所需要的页面，这里新建 3 个 jsp 页面：login.jsp、login_success.jsp 和 login-failure.jsp。login.jsp 页面接受用户的输入，如果判断为合法用户且密码正确，则跳转到 login_success.jsp，否则跳转到 login_failure.jsp。

（5）编辑 login.jsp 内容如下：

```
<%@ page language="java" contentType="text/html; charset=utf-8"
   pageEncoding="utf-8"%>

<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>登录</title>
</head>
<body>
<center>
请用您的用户名和密码登录
</center>
<html:form action="login.do" method="post">
    用户名：<html:text size="15" property="username" />
    <br>
    密码：<html:password size="15" property="password" />
    <br>
    <html:submit property="login" value="Login" />

</html:form>

</body>
</html>
```

① 编辑 login_success.jsp，内容如下：

```
<%@ page language="java" contentType="text/html; charset= ISO-8859-1"
pageEncoding=" ISO-8859-1"%>
<!—登录成功页面，显示登录成功信息-->
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>登录成功</title>
</head>
<body>
登录成功
</body>
</html>
```

② 编辑 login_faiure.jsp 文件，内容如下：

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding=" ISO-8859-1"%>
<!—登录失败页面，显示登录失败信息-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>登录失败</title>
</head>
<body>
登录失败
</body>
</html>
```

（6）建立 Action 类和 ActionForm 类

① Action 类（LoginAction.java），内容如下：

```
import org.apache.struts.action.*;
import javax.servlet.http.*;
import java.io.IOException;
import javax.servlet.ServletException;


public class LoginAction extends Action {

    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest req, HttpServletResponse res)
            throws IOException, ServletException {

            return mapping.findForward("login");

    }

}
```

② Form 类（LoginForm.java），内容如下：

```
import org.apache.struts.action.*;

public class LoginForm extends ActionForm {
    private static final long serialVersionUID = 1L;

    private String username;
    private String password;

    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }

}
```

（7）编写 Struts-config.xml 配置文件，内容如下：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
        "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<!--
    NOTE: If you have a generator tool to create the corresponding Java classes
    for you, you could include the details in the "form-bean" declarations.
    Otherwise, you would only define the "form-bean" element itself, with
    the corresponding "name" and "type" attributes, as shown here.
-->

<struts-config>

    <form-beans>
        <form-bean name="loginForm" type="LoginForm" />
    </form-beans>


    <global-exceptions>

    </global-exceptions>
```

```
    <!-- -------------- Global Forward Definitions ------------------->
    <!-----------------Action Mapping Definitions ------------------->

    <action-mappings>
        <action name="loginForm" path="/login" type="LoginAction">
            <forward name="login" path="/login.jsp"></forward>
            <forward name="success" path="/login_success.jsp"></forward>
            <forward name="failure" path="/login_failure.jsp"></forward>
        </action>
    </action-mappings>

    <!-----------------Controller Configuration ---------------------->
    <!-----------------Message Resources Definitions ------------------>
    <!---------------- Plug Ins Configuration ------------------------->

</struts-config>
```

现在重新启动 Tomcat 服务器（每次修改了 struts-config.xml 等配置文件，都需要重启服务器才能生效），访问 http://localhost:8080/Example1/login.do，如果能看到接受输入的页面，就说明 Struts 已经跑通了。

（8）接下来添加 Hibernate 所需要的包，主要是 Hibernate3.jar 和 lib 目录下的.jar 包，把它们导入到 WEB-INF/lib 目录下，以支持 Hibernate。

（9）将 MySQL 的 JDBC Driver（mysql-connector-java-5.0.7-bin.jar）导入到 WEB-INF/lib 目录下。用 Query Browser 建立表 USER，共 3 个字段——CID:INTEGER、USER-NAME:VARCHAR(45)和 PASSWORD:VARCHAR(45)，并插入一些数据。

（10）建立 Hibernate 映射，新建 User 类：

```
public class User {


   private String username;
   private String password;

   private int id;

   public int getId() {
       return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getPassword() {
      return password;
   }
   public String getUsername() {
```

```
        return username;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public void setUsername(String username) {
        this.username = username;
    }

}
```

（11）编写 USER 表的映射文件 User.hbm.xml，内容如下：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>
    <class name="User" table="USER">
        <id name="id" column="ID">
            <generator class="increment" />
        </id>
        <property name="username" column="USERNAME" />
        <property name="password" column="PASSWORD" />
    </class>
</hibernate-mapping>
```

（12）编写 hibernate.cfg.xml 文件，内容如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE hibernate-configuration
    PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-2.0.dtd">

<hibernate-configuration>

    <session-factory name="java:/hibernate/HibernateFactory">

        <property name="show_sql">true</property>
        <property name="connection.driver_class">org.gjt.mm.mysql.Driver</
        property>
        <property name="connection.url">jdbc:mysql://localhost/hibernate</
        property>
        <property name="connection.username">root</property>
        <property name="connection.password"></property>
        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
```

```
        </property>

        <mapping resource="User.hbm.xml" />

    </session-factory>

</hibernate-configuration>
```

（13）修改 LoginAction 以进行 Hibernate 操作，并实现所需要的逻辑，内容如下：

```
import org.apache.struts.action.*;
import javax.servlet.http.*;
import java.io.IOException;
import javax.servlet.ServletException;
import java.util.ArrayList;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class LoginAction extends Action {

    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest req, HttpServletResponse res)
            throws IOException, ServletException {

        LoginForm loginForm = (LoginForm) form;

        //Get the username and password from user input
        String username = loginForm.getUsername();
        String password = loginForm.getPassword();

        try {
            SessionFactory sf = new Configuration().configure()
                    .buildSessionFactory();
            Session session = sf.openSession();
            Transaction tx = session.beginTransaction();


            //Select the users from database with the username
            String sqlQuery = "select u from User u where u.username='"
                    + username + "'";
            Query lQuery = session.createQuery(sqlQuery);
            ArrayList userList = (ArrayList) lQuery.list();
            tx.commit();
            session.close();

            User user = new User();

            //There is such a user
            if ((null != userList) && (userList.size() > 0)) {
```

```
                user = (User) userList.get(0);
                if (!user.getPassword().equals(password)) {
                    return mapping.findForward("failure");
                }
            } else {
                //There is no such user
                return mapping.findForward("failure");
            }

        } catch (HibernateException e) {
            e.printStackTrace();
            return mapping.findForward("failure");
        }

        //Successfully login
        return mapping.findForward("success");

    }

}
```

（14）Login_success.jsp 修改后如下：

```
<%@ page language="java" contentType="text/html; charset=utf-8"
   pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>登录成功</title>
</head>
<body>
欢迎你：<bean:write name="loginForm" property="username" />
</body>
</html>
```

（15）Login_failure.jsp 修改后如下：

```
<%@ page language="java" contentType="text/html; charset=utf-8"
   pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>登录失败</title>
</head>
```

```
<body>
错误的用户名或密码: <a href="javascript:window.location('/Example1/login.
jsp')">Return</a>
</body>
</html>
```

现在访问 http://localhost:8080/Example1/login.jsp 即可看到成果了。

7.4　诊断 Connector/J 方面的问题

在本节总结了在使用 Connector/J 时遇到的一些常见问题，并提供了一些解决办法。

7.4.1　关于授权问题

尝试用 MySQL Connector/J 连接到数据库时，遇到下述异常：

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

使用 MySQL 命令行客户端时，连接良好。

解决办法：

MySQL Connector/J 必须使用 TCP/IP 套接字来连接 MySQL，原因在于 Java 不支持 UNIX Domain 套接字。因此，当 MySQL Connector/J 连接到 MySQL 时，MySQL 服务器的安全管理器将使用其授权表判断是否允许连接。必须添加授权才能允许该操作。下面给出了一个执行该操作的示例。

从 MySQL 命令行客户端以能够授权的用户身份登录，并发出下述命令：

```
GRANT ALL PRIVILEGES ON [dbname].* to
          '[user]'@'[hostname]' identified by
          '[password]'
```

用数据库名称替换[dbname]，用用户名替换[user]，用 MySQL Connector/J 将连接的主机替换[hostname]，并用打算使用的密码替换[password]。注意，对于从本地主机进行连接的主机名部分，RedHat Linux 将失败。在这种情况下，对于[hostname]值，需要使用 localhost.localdomain。随后，发出 FLUSH PRIVILEGES 命令。

7.4.2　SQLException，无法连接到 MySQL 服务器

当试图在 Java 程序或应用程序中使用 MySQL Connector/J 时，有时会遇到类似下面的异常。

SQLException：无法连接到 host:3306 上的 MySQL 服务器。

在尝试连接的机器/端口上是否有正在运行的 MySQL 服务器？

```
      (java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

解决办法：

这种情况可能是正在运行 Applet，MySQL 服务器是采用“--skip-networking”选项集安装的，或许是因为 MySQL 服务器位于防火墙之后。Applet 仅能使网络连接返回运行 Web 服务器的机器，该 Web 服务器提供了用于 Applet 的.class 文件。这意味着，要想使其工作，MySQL 必须运行在相同的机器上（或必须使某类端口重定向），且无法通过本地文件系统来测试 Java 程序，必须将它们放在 Web 服务器上。

MySQL Connector/J 仅能使用 TCP/IP 与 MySQL 进行通信，这是因为 Java 不支持 UNIX 域套接字。如果 MySQL 是用“--skip-networking”标志启动的，或采用了防火墙，TCP/IP 与 MySQL 的通信可能会受到影响。

如果 MySQL 是用“--skip-networking”选项集启动的（例如 MySQL 服务器的 Debian Linux 包即用于该目的），需要在文件/etc/mysql/my.cnf 或/etc/my.cnf 中将其注释掉。当然，my.cnf 文件也可能位于 MySQL 服务器的“data”目录下或其他地方（取决于系统中 MySQL 的编译方式）。MySQL AB 创建的二进制文件在不断地查找/etc/my.cnf 和[datadir]/my.cnf。如果为 MySQL 服务器部署了防火墙，需要对防火墙进行配置，允许从运行 Java 代码的主机在 MySQL 监听的端口上（默认为 3306）建立与 MySQL 服务器的 TCP/IP 连接。

7.4.3　结果集不可更新

在尝试使用 JDBC-2.0 可更新结果集，但遇到异常，说明结果集不可更新。

解决办法：

由于 MySQL 没有行 ID，MySQL Connector/J 仅能更新来自查询且位于有至少一个主键的表上的结果集，查询必须选择所有的主键，而且查询只能作用在一个表上（即不存在联合）。在 JDBC 规范中给出了这方面的介绍。

7.4.4　如何通报缺陷和问题

通报缺陷的正常地址是 http://bugs.mysql.com，这是 MySQL 的官方缺陷数据库的地址。这是一个公共数据库，任何人都能浏览它并进行相应的搜索。如果已登录到系统，也应能输入新的报告。

如果发现 MySQL 中存在敏感的安全缺陷，也可以发送电子邮件至 security@mysql.com。

7.5　Connector/J 的版本

Connector/J 是所谓的第四类驱动程序，意思是说它的全部功能由 Java 实现。Connector/J

要求系统必须安装 JDBC 2.0 或更高的版本。就目前而言，比较常见的 Connector/J 有 3 种：

（1）Connector/J 3.0，这是为 MySQL 3.23 和 4.0 版本开发的 Connector/J 版本。

（2）Connector/J 3.1，可以配合目前流行的所有 MySQL 版本工作。这个版本能够支持 MySQL 4.1 版本的新增功能，如新的密码身份验证机制、Unicode 字符集、预处理语句等。

（3）Connector/J 3.2，这是 2005 年 3 月推出的 Alpha 测试版本。这个版本能够支持 MySQL 5.0 版的新增功能，如光标函数等。建议在 Connector/J 3.2 正式发布之前，应该尽量选用 Connector/J 3.1 版。

7.6　小　　结

在本章，应该把重点放在 JDBC 的编程以及在 J2EE 中如何结合流行的框架进行数据库的开发上。要求扎实掌握 JDBC 编程的基本步骤和数据库的 DML 操作，对 J2EE 的开源框架，也应该有一定的了解。

第 3 篇



打好了开发的基础，还需要有项目来进行实战操练。在本篇中将给出 4 个完整的项目案例——用户管理系统、CASE 支撑系统、文件管理系统、教务管理系统。学完本篇后读者应能够运用已掌握的 MySQL 和 Java 程序设计的知识进行综合练习，本篇可以全面巩固读者的知识，培养读者解决实际问题的能力，从而达到学以致用的目的。

第 8 章　用户管理系统案例

用户管理是所有管理系统的基础，本章通过一个项目“用户管理系统”的设计与开发，描述了在 Web 中间件 Tomcat 环境下，如何设计一个比较通用的用户与权限管理系统。

8.1　系统需求分析

用户与权限是一个比较复杂的问题，针对不同的应用，需要根据项目的实际情况和具体架构，在维护性、灵活性、完整性等几个方案之间比较权衡，选择符合项目实际的方案。

8.1.1　需求概述

本系统要达到以下目标：

（1）直观。因为系统最终会由用户来维护，权限分配的直观且容易理解，显得比较重要，系统实现角色（即权限组）的继承，除了功能的必需，更主要的就是因为它足够直观。

（2）简单。包括概念上的简单和功能上的简单，不考虑用户组。

（3）可移植。系统扩展性要强，要便于移植到不同的系统中。

本系统默认有一个系统管理员用户。系统管理员的工作有：

① 增加、删除、修改和查询部门。

② 增加、删除、修改和查询用户。

③ 增加、删除、修改和查询角色。

④ 组合操作权限分配给角色。

⑤ 将角色分配给用户。

User（用户）与 Role（角色，拥有一定数量的权限）相关，用户仅仅是纯粹的用户，权限是被分离出去了的。User 是不能与 Privilege（权限）直接相关的，User 要拥有对某种资源的权限，必须通过 Role 去关联。

用户管理视图如图 8-1 所示。

用户管理业务流程图如图 8-2 所示。

根据上面的需求，画出用户管理系统的用例图，如图 8-3 所示。

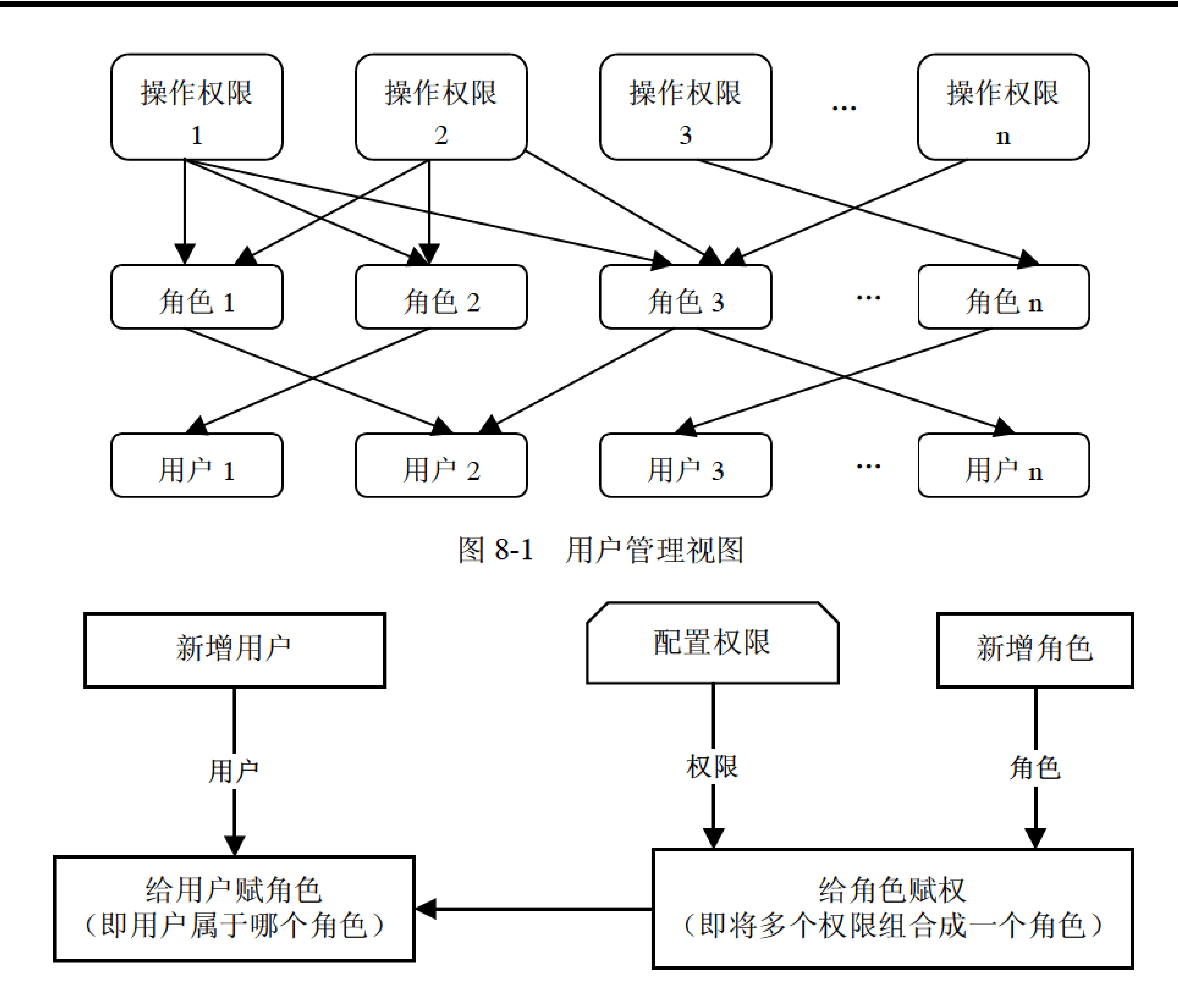


图 8-1　用户管理视图

图 8-2　用户管理业务流程图

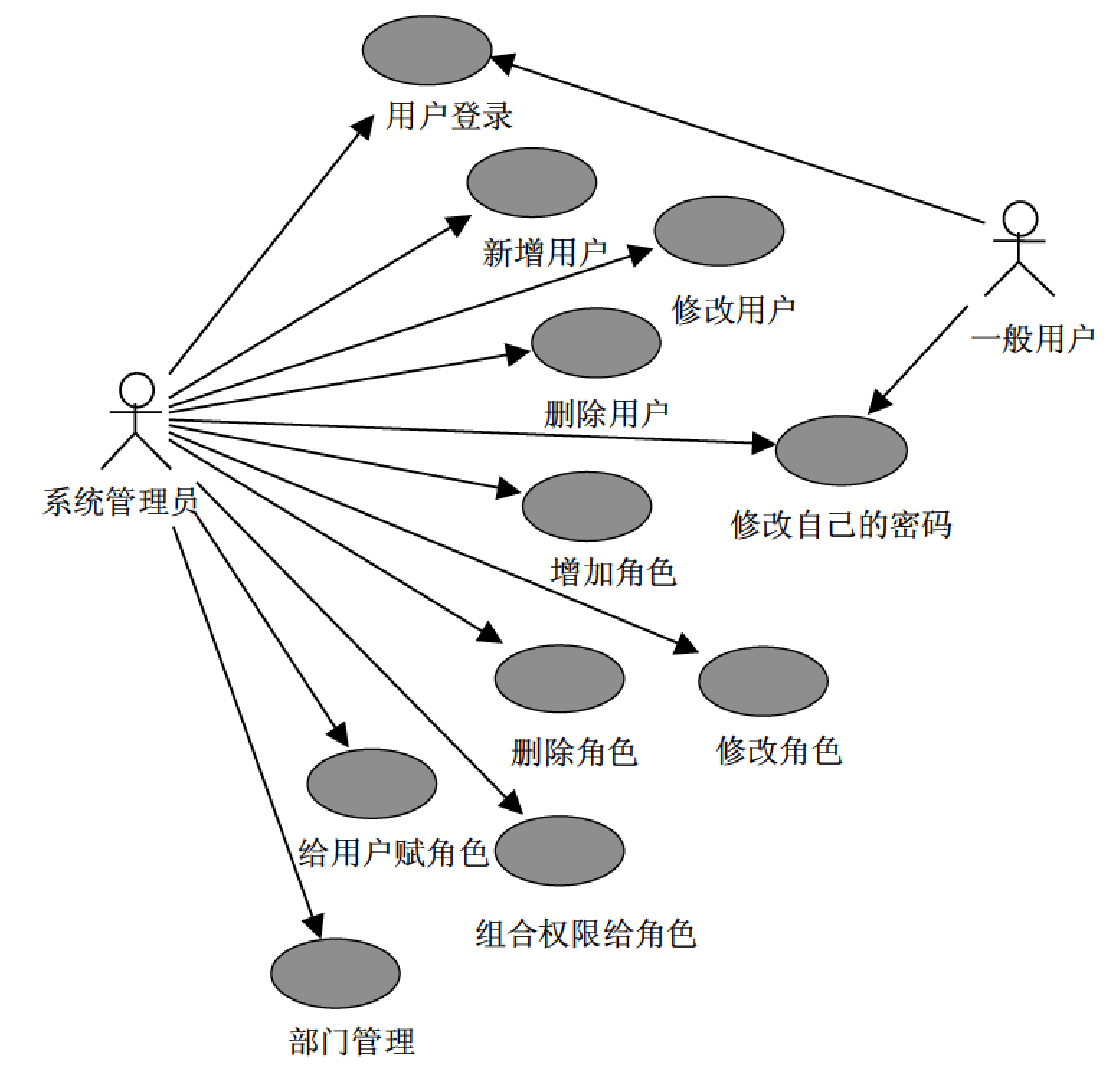


图 8-3　系统用例图

8.1.2　系统功能描述

系统功能主要包括用户管理、角色管理。

1．用户管理

❑　新增用户

增加用户时，系统中将自动增加一条相应的用户。带“*”号的为必填项。“是否禁用”分为“禁用”和“不禁用”，选择“禁用”表示系统将停止该用户进入系统的权限。“用户性质”分为“一般用户”和“管理员”两种，如图 8-4 所示。

选择所属部门，将出现选择部门界面，如图 8-5 所示。



图 8-4　新增用户界面



图 8-5　选择部门界面

❑　修改用户

提供修改用户的基本属性，如图 8-6 所示。



图 8-6　修改用户界面

❑　删除用户

系统将该用户直接删除，并且将该用户从其归属的权限删除。

❑　用户赋权

给选中的用户赋权，角色可以多选，如图 8-7 所示。

❑　权限查看

显示用户具有的所有权限，如图 8-7 所示。



图 8-7　用户赋权界面

❑　密码管理

用户可以修改密码。主要属性为用户原密码、用户新密码、用户新密码确认，如图 8-8 所示。

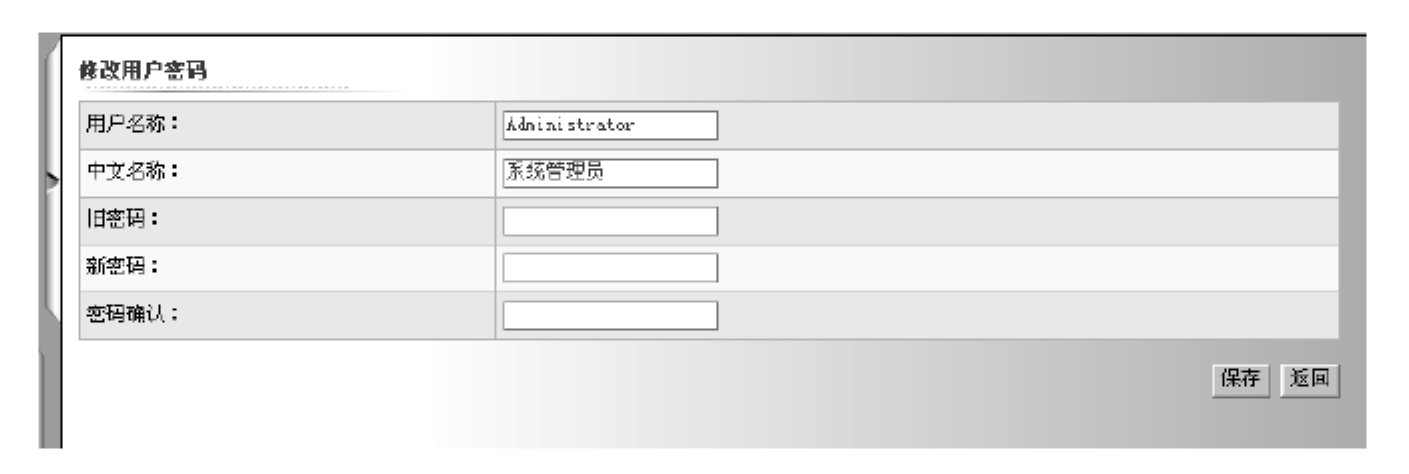


图 8-8　用户修改密码界面

❑　用户查看

主要查看已经创建的用户，如图 8-9 所示。



图 8-9　用户查看界面

2. 角色管理

❑　新增角色

增加角色，输入图 8-10 中角色的基本属性信息，系统中将产生相应的一条新记录。

角色的基本属性包括角色 ID（系统自动取得一个唯一值）、角色名称、角色描述、角

色性质。

可以自己组合选择系统定义的基本权限。

新增角色

角色名称：	
角色描述：	
角色性质：	系统管理员
□选择全部权限	

权限名称	权限名称	权限名称
□设待办事项为已读	□查看我的桌面内容	□显示员工列表权
□社区人员查询全部	□社区人员按条件查询	□社区人员查询按ID查询
□增加用户权	□删除用户权	□预增加用户权
□删除用户角色权	□用户密码修改权	□预修改用户对应角色权
□修改用户对应角色权	□预修改用户权	□修改用户权
□显示用户对应角色列表权	□显示用户列表权	□显示所有用户权
□增加角色权	□预增加角色权	□删除角色权
□修改角色权	□修改角色对应用户权	□预修改角色对应用户权
□预修改角色权	□显示角色列表权	□显示角色对应权限列表权
□增加用户角色权	□增加工作时间权	□删除工作时间权

图 8-10　新增与修改角色界面

❑　修改角色

系统允许对角色的属性信息进行修改操作。

允许增减角色中所包含的功能单元（权限），如图 8-10 所示。

❑　删除角色

系统可以删除所建的角色。

❑　用户赋权

将选中的一个角色赋权给指定的用户，用户可以多选，如图 8-11 所示。



图 8-11　修改角色授权用户

8.2　系统总体架构

本系统采用 J2EE 的三层结构，分为表现层、业务逻辑层和数据服务层。三层体系将业务规则、数据访问等工作放到中间层处理，客户端不直接与数据库交互，而是通过控制器与中间层建立连接，再由中间层与数据库交互。

系统总体架构如图 8-12 所示。

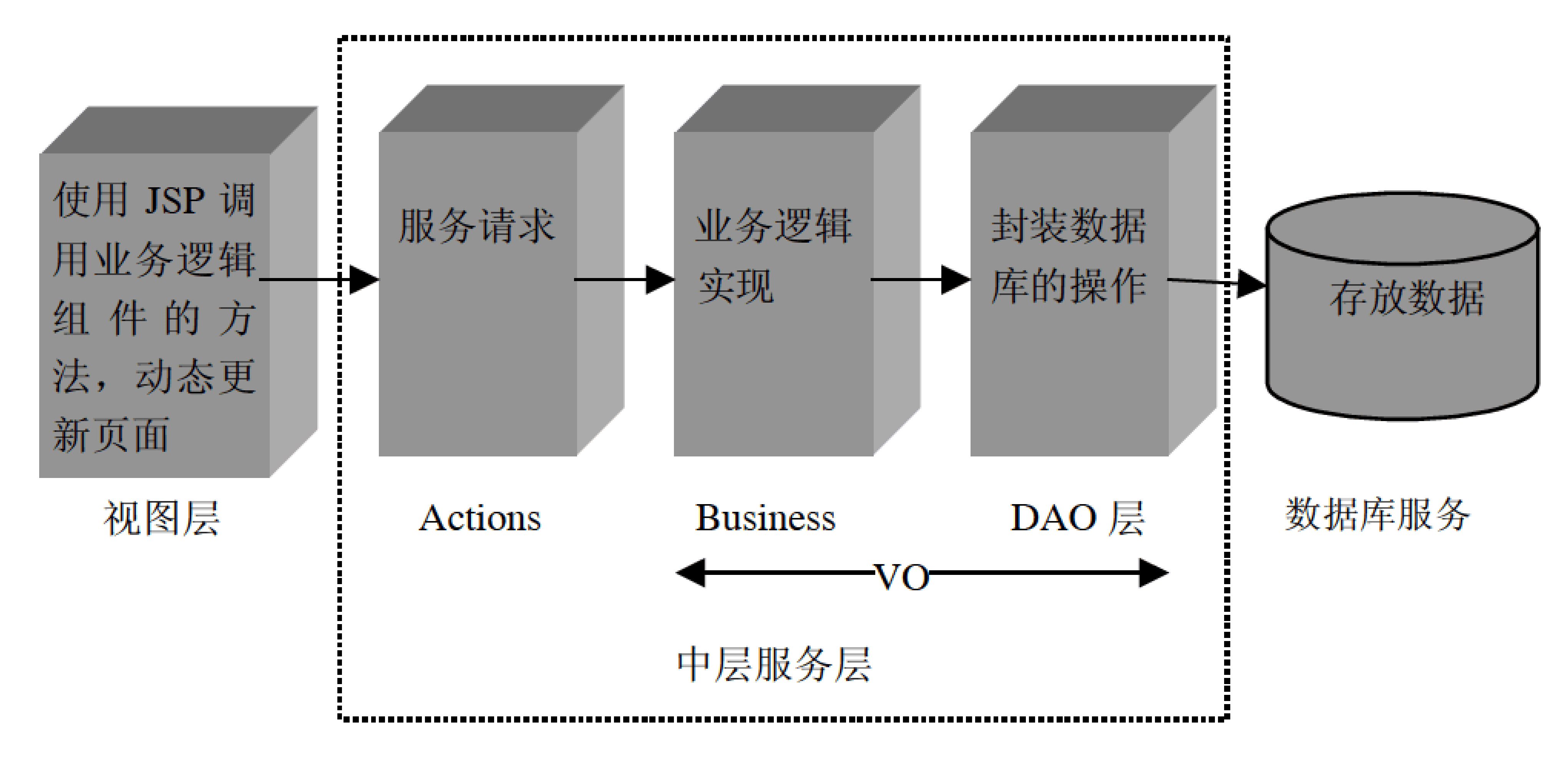


图 8-12　系统的总体设计图

其中：

- Business（业务逻辑）：负责实现业务逻辑，对 DAO 对象进行封装。
- DAO（数据访问对象）：负责与数据库的交互，封装了数据的增加、删除、修改、查询等操作。
- VO（数值对象）：负责与数据库基表的映射。

8.3　数据库设计

数据库设计需要包括业务实体设计和数据模型设计。

8.3.1　业务实体设计

一个系统的业务实体在计算机内存中表现为实体域对象，在数据库中表现为关系数据，实现业务实体包括以下内容：

- 设计域模型，创建域模型实体对象。
- 设计关系数据模型。
- 创建对象—关系映射文件。

根据前面的系统需求分析，本系统中可以抽象出来的业务实体包括用户、部门、角色、权限等。这几个实体相互进行交互，下面介绍这些实体模型的定义。

- 用户。代表一个可操作的用户实体，主要属性有用户名称、用户描述、用户密码、用户启用标志、用户所在的部门等。
- 部门。代表一个部门实体，主要属性包括部门名称、上级部门、类型等。
- 角色。代表一个角色实体，主要属性有角色名称、角色描述、角色性质等。

- 权限。代表一个权限实体，主要属性有权限名称、权限描述、类型等。

8.3.2　数据模型设计

根据以上的关系来构造系统的数据模型。

下面来建立各个数据模型。具体设计情况如下各表所示。

1．用户信息表（p_user）

表 8-1 主要用来保存用户的信息，包括用户名称、密码、用户描述等。

表 8-1　用户信息表

字 段 名	中 文 含 义	字 段 类 型	长 度	备 注
USER_ID	用户 ID	INT	9	主键
USER_NAME	用户名称	VARCHAR	20	
USER_DESCRIPTION	用户描述	VARCHAR	100	
PASSWORD	密码	VARCHAR	20	
USER_ENABLED	用户启用标志	VARCHAR	1	
ADMIN	是否管理员	VARCHAR	1	
DEPARTMENT_ID	用户所属部门	INT	9	
ON_LINE	是否在线	VARCHAR	1	
REMARK	备注	VARCHAR	80	

2．权限信息表（p_popedom）

表 8-2 主要用来保存权限的信息，包括权限名称、权限描述、类型等。

表 8-2　权限信息表

字 段 名	中 文 含 义	字 段 类 型	长 度	备 注
POPEDOM_NAME	权限名称	VARCHAR	200	唯一索引，主键
POPEDOM_ALIAS	权限描述	VARCHAR	200	
POPEDOM_TYPE	类型	VARCHAR	1	S 为权限，R 为角色，P 为管理员角色
POPEDOM_SORT	权限排序	VARCHAR	10	
POPEDOM_ACTION	关联动作	VARCHAR	200	Web 应用程序将 Action 对应权限

3．角色权限关系表（p_role_popedom）

表 8-3 主要用来保存角色权限关系的信息，包括角色名称、权限名称。

表 8-3　角色权限关系表

字 段 名	中 文 含 义	字 段 类 型	长 度	备 注
ROLE_NAME	角色名称	VARCHAR	20	组合主键
POPEDOM_NAME	权限名称	VARCHAR	20	组合主键

4. 用户角色关系表（p_user_role）

表 8-4 主要用来保存用户角色关系的信息，包括用户名称、角色名称等。

表 8-4　用户角色关系表

字　段　名	中 文 含 义	字 段 类 型	长　　度	备　　注
USER_NAME	用户名称	VARCHAR	20	组合主键
ROLE_NAME	角色名称	VARCHAR	100	组合主键

5. 部门信息表（eoa_dept）

表 8-5 主要用来保存部门的信息，包括用户名称、角色名称等。

表 8-5　部门信息表

字　段　名	中 文 含 义	字 段 类 型	长　　度	备　　注
DEPT_ID	部门 ID	INT	9	主键
DEPT_NAME	部门名称	VARCHAR	100	
DEPT_CODE	部门编号	VARCHAR	50	
PARENT_ID	上级部门 ID	INT	9	
DEPT_TYPE	部门类型	VARCHAR	20	
ENABLED	启用标志	VARCHAR	1	
DESCRIPTION	部门描述	VARCHAR	100	
QUEU	排序号	INT	4	
REMARK	备注	VARCHAR	80	

由于考虑比较通用性和可移植性，不采用自增量字段或者序列生成器以及存储过程或者函数、触发器，仅用了视图。视图设计如下：

```
CREATE OR REPLACE VIEW P_USER_POPEDOM_VIEW AS
SELECT USER_NAME,POPEDOM_NAME
FROM P_USER_ROLE A,P_ROLE_POPEDOM B
WHERE A.ROLE_NAME= B.ROLE_NAME
```

8.4　系统详细设计

系统详细设计包括界面设计、逻辑主线、系统视图设计、系统包设计、数据库访问连接设计和业务层设计。

8.4.1　界面设计

在实际软件项目开发中，界面设计是需求分析之后首先要做的一件事，通过设计的界面来和用户进行沟通，从而达成一致的需求。

在本系统的界面设计中，使用 CSS 进行界面设计。CSS 文件在本书提供的源代码中，

在工程文件夹\USER\css 中有多个 CSS 样式单，这些样式单文件管理页面是最通用的显示效果。当然，读者可以设计自己的 CSS 样式单文件。

\USER\images 路径中有多个必须的 GIF 文件、JPG 图片文件。当然，读者也可以设计自己喜欢的图片。

具体界面设计参见 8.1.2 节中的各个设计页面。

8.4.2　逻辑主线

一个 Java Web 应用所需要的一个核心文件就是 web.xml。该文件控制整个应用的行为方式和方法，这是通过各种命令参数的配置来实现的，这些参数在服务启动时自动加载。web.xml 中的元素和 Tomcat 容器完全独立，它是 Web 应用的配置文件。

这里主要介绍贯穿整个系统的 Web 服务基本配置文件 web.xml，其位于\Tomcat 5.5\webapps\USER\WEB-INF 下。

```
<?xml version="1.0" encoding="UTF-8"?>  <!-- 指定版本和文字编码 -->
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd"> <!-- 指定 DTD 文档的位置
-->
<web-app>
  <display-name>用户管理系统 V1.0</display-name>
                <!-- 标记特定的 Web 应用的名称 -->
  <filter>
       <!--过滤器定义。将一个名字与一个实现 Javax.servlet.Filter 接口类关联。
       这里定义一个过滤器，名为 MainFilter。实现这个过滤器的类是
       com.dfkj.web.MainFilter ->
    <filter-name>com.dfkj.web.MainFilter</filter-name>
    <filter-class>com.dfkj.web.MainFilter</filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>GBK</param-value>
    </init-param>
    <init-param>
      <param-name>debug_flag</param-name>
      <param-value>on</param-value>
    </init-param>
  </filter>
  <filter>  <!-- 权限过滤器定义 -->
    <filter-name>com.dfkj.web.PowerFilter</filter-name>
    <filter-class>com.dfkj.web.PowerFilter</filter-class>
    <init-param>
      <param-name>flag</param-name>
      <param-value>off</param-value>  <!--off 表示权限有效，on 无效 -->
    </init-param>
  </filter>
  <filter-mapping>
    <!—主过滤器映射：一旦命名了一个过滤器，就要利用 filter-mapping 元素把它
```

```
与一个或多个 Servlet 或 JSP 页面相关联 -->
    <filter-name>com.dfkj.web.MainFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <filter-mapping>  <!--权限过滤器映射-->
    <filter-name>com.dfkj.web.PowerFilter</filter-name>
    <url-pattern>/MainController.do</url-pattern>
  </filter-mapping>
  <servlet>
    <!--Servlet 定义：在向 Servlet 或 JSP 页面制定初始化参数或定制 URL 时，必
      须首先命名 Servlet 或 JSP 页面。Servlet 元素就是用来完成此项任务的-->
    <servlet-name>DBServ</servlet-name>
    <display-name>DBServ</display-name>
    <description>用于数据库连接初始化</description>
    <servlet-class>com.dfkj.db.DBServ</servlet-class>
    <load-on-startup>3</load-on-startup>
  </servlet>
  <servlet>
    <!--数据库初始化文件，位于\USER\WEB-INF\classes-->
    <servlet-name>DbInitConfig</servlet-name>
    <servlet-class>com.dfkj.db.DbInitConfig</servlet-class>
    <init-param>
      <param-name>DbInitFile</param-name>
      <param-value>WEB-INF/classes/adapter.xml</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>Log4jInit</servlet-name>
    <display-name>Log4jInit</display-name>
    <description>Log4j 初始化</description>
    <servlet-class>com.dfkj.log.Log4JInit</servlet-class>
    <init-param>
      <param-name>Log4jInitFile</param-name>
      <param-value>WEB-INF/classes/Log4j.properties</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
        <!--这里定义了一个名为 MainController 的 Servlet，实现这个 Servlet 的类
        是 com.dfkj.web.MainController。 -->
    <servlet-name>MainController</servlet-name>
    <display-name>MainController</display-name>
    <description>用于改变字符集</description>
    <servlet-class>com.dfkj.web.MainController</servlet-class>
    <load-on-startup>4</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>action</servlet-name>
```

```
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>validate</param-name>
      <param-value>true</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>treebuilders</param-name>
     <param-value>org.apache.webapp.admin.DepartTreeBuilder</param-value>
    </init-param>
    <init-param>
      <param-name>application</param-name>
      <param-value>ApplicationResources</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>debugjsp</servlet-name>
    <description>编译 JSP 时增加调试信息</description>
    <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
    <init-param>
      <param-name>classdebuginfo</param-name>
      <param-value>true</param-value>
    </init-param>
    <load-on-startup>3</load-on-startup>
  </servlet>
  <servlet-mapping>
    <!-- Servlet 映射：服务器一般为 Servlet 提供一个默认的
    URL：http://host/webAppPrefix/servlet/ServletName。但常常会更改这个
    URL，以便 Servlet 可以访问初始化参数或更容易地处理相对 URL 时，使用
     servlet-mapping 元素-->
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>debugjsp</servlet-name>
    <url-pattern>*.jsp</url-pattern>
  </servlet-mapping>
```

```
<servlet-mapping>
  <servlet-name>MainController</servlet-name>
  <url-pattern>/MainController.do</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>MainController</servlet-name>
  <url-pattern>/LogoutController.do</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>RegionOutput</servlet-name>
  <url-pattern>/RegionOutput</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>Log4jInit</servlet-name>
  <url-pattern>/servlet/Log4jInit</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>DBServ</servlet-name>
  <url-pattern>/servlet/com.dfkj.db.DBServ</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>DbInitConfig</servlet-name>
  <url-pattern>/servlet/com.dfkj.db.DbInitConfig</url-pattern>
</servlet-mapping>
<session-config>  <!-- 控制会话超时。这里 Session 可以保持不活动状态的最长
                   时间为 60 秒，超过这一时间，Servlet 容器将把它作为无效
                   Session 处理-->
  <session-timeout>60</session-timeout>
</session-config>
<welcome-file-list>
   <!-- 指定欢迎文件页：指示服务器在收到引用一个目录名而不是文件名的 URL
   时，显示哪个文件作为欢迎页。这里显示 login.jsp -->
  <welcome-file>login.jsp</welcome-file>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
</welcome-file-list>
<taglib>
  <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>
<taglib>
```

```
    <taglib-uri>/WEB-INF/struts-template.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-template.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/struts</taglib-uri>
    <taglib-location>/WEB-INF/lib/struts.jar</taglib-location>
  </taglib>
  <taglib>   <!--定位 TLD：taglib 元素对标记库描述符文件指定别名-->
    <taglib-uri>dfkj</taglib-uri>
    <taglib-location>/WEB-INF/tlds/dfkj.tld</taglib-location>
  </taglib>
</web-app>
```

8.4.3　系统中的视图设计

在本系统的开发中，首先要完成系统中所有视图的创建。

- 系统主视图及相关视图（如表 8-6 所示）

表 8-6　系统主视图及相关视图表

视图（jsp）	描　　述
blank.jsp	空白页面
Bottomframe.jsp	底部框架页面
login.jsp	登录页面
logout.jsp	注销页面
main.jsp	登录后的上、中、下框架页面
page_error.jsp	错误提示页面
session_error.jsp	会话错误页面

- 用户管理部分视图（如表 8-7 所示）

表 8-7　用户管理部分视图表

视图（jsp）	描　　述
departmentList.jsp	部门列表页面
pwdDone.jsp	密码修改完毕页面
pwdError.jsp	密码不正确提示页面
pwdModify.jsp	密码修改页面
roleAdd.jsp	增加角色页面
roleList.jsp	角色列表页面
roleModify.jsp	修改角色页面
roleuserModify.jsp	修改角色对应的用户页面

续表

视图（jsp）	描　　述
userAdd.jsp	增加用户页面
userChangePassword.jsp	用户修改密码页面
userList.jsp	用户列表页面
userModify.jsp	修改用户页面
userroleModify.jsp	修改用户角色页面

8.4.4　系统中的包设计

系统中的包设计如图 8-13 所示。

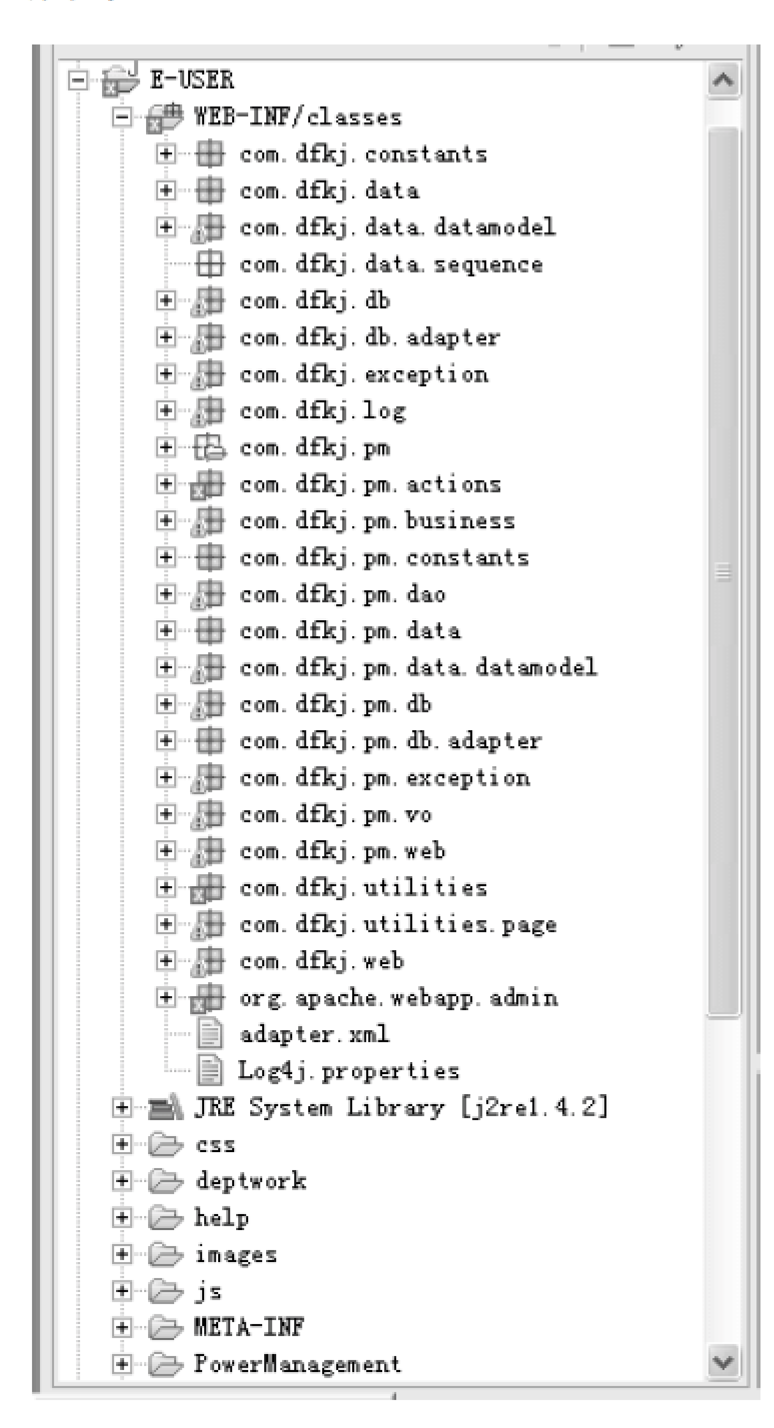


图 8-13　系统包结构图

下面对每个包进行简要说明，如表 8-8 所示。

表 8-8　包说明表

包	类	描　　述
USER\WEB-INF\classes\com\dfkj\pm\actions	所有请求的服务类	
USER\WEB-INF\classes\com\dfkj\pm\business	业务逻辑/流程的实现类	
USER\WEB-INF\classes\com\dfkj\pm\common	存放通用的类	
USER\WEB-INF\classes\com\dfkj\pm\constants	本系统所用到的常量类	
USER\WEB-INF\classes\com\dfkj\pm\dao	访问数据库的操作（增加、删除、查询）类	

续表

包	类	描　述
USER\WEB-INF\classes\com\dfkj\pm\vo	与数据库基表的映射，即数值对象类	
USERWEB-INF\classes\com\dfkj\web	Iaction.class	实现 Iaction 接口，所有请求服务类的接口
	LoginInfor.class	登录信息
	MainController.class	主控制器
	MainFilter.class	初始化字符集过滤器
	PowerFilter.class	权限控制
	RequestProcessor.class	对主控制器补充的一些类
	WebParam.class	统一定义参数名称
USER\WEB-INF\classes\com\dfkj\pm	权限部分类	
USER\WEB-INF\classes\com\dfkj\log	日志类	
USER\WEB-INF\classes\com\dfkj\data	序列生成器等类	
USER\WEB-INF\classes\com\dfkj\exception	程序中定义的异常类	
USER\WEB-INF\classes\com\dfkj\db	数据库操作封装、datamodel 等类	

8.4.5　数据库的访问连接设计

在本系统中，对数据库的访问连接是通过一个配置文件 adapter.xml 来设置的。

对于具有高数据访问量的应用来说，一个好的策略就是管理一个连接池。通俗地说，就是将每次创建的数据库连接放在一个“池”里，并且在连接使用完成时并不急于关闭这个连接。当应用程序需要调用一个数据库连接时，数据库相关的接口通过返回一个重用数据库连接来代替重新创建一个数据库连接，只在没有可用的数据库连接时，才重新创建一个。通过这种方式，应用程序可以减少对数据库连接的操作，尤其在多层环境中，多个客户端可以通过共享少量的物理数据库连接来满足系统需求。

数据库连接的适配器等类位于\USER\WEB-INF\classes\com\dfkj\db 包中。下面是 adapter.xml 文件的代码设计，该文件位于\USER\WEB-INF\classes 下。

❑　adapter.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) -->
<adapter>
    <!--连接池开关，使用连接池则 connectionType 项为 Pool,
    否则可以为其他任意值-->
    <connectionType>USER</connectionType>
```

```
    <!--单独连接的驱动选择，当 connectionType 不是 Pool 时，
driverAdapter 在哪个 driversName 前面，则使用该 driver-->
    <driverAdapter>Oracle</driverAdapter> <!--直接连接数据库；
                                              这里连接 Oracle-->
    <driversName id="MYSQL">
        <userName>mysql</userName>
        <passWord>usemysql</passWord>
        <url>jdbc:mysql://171.71.10.2:3307/USER</url>
        <driver>org.gjt.mm.mysql.Driver</driver>
        <adapterName>MysqlDbAdapter</adapterName>
    </driversName>
    <driversName id="Oracle">
        <userName>eoa</userName>
        <passWord>eoa</passWord>
        <url>jdbc:oracle:thin:@127.0.0.1:1521:orcl</url>
        <driver>oracle.jdbc.driver.OracleDriver</driver>
        <adapterName>OracleDbAdapter</adapterName>
    </driversName>
    <driversName id="ODBC">
        <userName>eoa</userName>
        <passWord>eoa</passWord>
        <url>jdbc:oracle:thin:@dell-2500:1521:ORA8</url>
        <driver>sun.jdbc.odbc.JdbcODBCDriver</driver>
        <adapterName>OdbcDbAdapter</adapterName>
    </driversName>

    <!--连接池驱动选择，当 connectionType 为 Pool 时，poolAdapter 在哪个
        driversName 前面就使用该 driver（需要在 tomcat 中配置好连接池）-->
    <poolAdapter>EoaPool</poolAdapter>
    <driversName id="MYSQLPOOL">
        <userName>mysql</userName>
        <passWord>usemysql</passWord>
        <url>jdbc:mysql://171.71.10.2:3307/USER</url>
        <driver>org.gjt.mm.mysql.Driver</driver>
        <adapterName>java:comp/env/mysqldb</adapterName>
    </driversName>
</adapter>
```

8.4.6　业务层设计

从整体上来说，可以把业务逻辑部分再分为两层：数据层逻辑和服务层逻辑。在实现业务层逻辑时，需要尽量保持这两个层次之间的松散耦合。这里分别就数据层和服务层的设计进行分析。

- ❑　数据层设计

在本系统中没有采用持久化管理，而是使用了 Java 的 JDBC 连接方式来实现。

因为 Java 的 JDBC 连接方式提供的方法很多，步骤也很多。为了实现数据库访问层和

逻辑业务层的尽量分离，通过 DAO 下的类来封装一些主要的数据库操作，例如事务操作、数据增加、数据删除、数据更新、数据各种查询等操作。

DAO 模式的实现至少需要 3 个部分：

- DAO 工厂类。
- DAO 接口。
- DAO 接口的实现类。

DAO 模式抽象出数据访问方式，业务逻辑组件不需要理会底层的数据库访问，而只专注于业务逻辑的实现。DAO 将数据访问集中在独立的一层，所有的数据访问都由 DAO 对象完成，DAO 分离了数据访问的实现与其他业务逻辑，使得系统更具有可维护性。

DAO 有助于提升系统的可移植性。独立的 DAO 层使得系统能在不同的数据库之间轻易切换，底层的数据库实现对于业务逻辑组件是透明的。数据库移植时仅影响 DAO，不同的数据库的切换不会影响业务逻辑组件，因而提高了系统的可复用性。

（1）DAO 工厂类

DAOFactory.class 是一个 DAO 工厂类。代码设计如下：

```
package com.dfkj.pm.dao;
import java.sql.Connection;
import java.util.Collection;
public class DAOFactory
{
   private DAOFactory()
   {
   }
   public static DAOFactory newInstance()
   {
      return new DAOFactory();
   }
   public IPPopedomDAO buildPPopedomDAO()  //权限
   {
      return new PPopedomDAOImpl();
   }
   public IPRolePopedomDAO buildPRolePopedomDAO() //角色对应权限
   {
      return new PRolePopedomDAOImpl();
   }
   public IPUserDAO buildPUserDAO()  //用户
   {
      return new PUserDAOImpl();
   }
   public IPUserPopedomViewDAO buildPUserPopedomViewDAO() //用户权限视图
   {
      return new PUserPopedomViewDAOImpl();
   }
   public IPUserRoleDAO buildPUserRoleDAO()  //用户对应的角色
   {
      return new PUserRoleDAOImpl();
```

```
    }
}
```

（2）DAO 接口的定义

本系统中 DAO 接口定义类，如表 8-9 所示。

表 8-9 DAO 接口定义类表

DAO 接口定义类	描　述
IPPopedomDAO	权限 DAO 定义接口
IPRolePopedomDAO	角色对应的权限 DAO 定义接口
IPUserDAO	用户 DAO 定义接口
IPUserPopedomViewDAO	用户权限视图 DAO 定义接口
IPUserRoleDAO	用户对应的角色 DAO 定义接口

下面以用户对应角色信息表为例进行设计说明。

IPUserRoleDAO（接口定义）类的代码设计如下：

```
package com.dfkj.pm.dao;
import java.sql.Connection;
import com.dfkj.pm.exception.*;
import com.dfkj.pm.vo.PUserRoleVO;
import java.util.Collection;

public interface IPUserRoleDAO
{
  public void insert(Connection transConn,PUserRoleVO pUserRoleVO)
    throws DaoException;                                  //增加记录
  public void update(Connection transConn,PUserRoleVO pUserRoleVO) throws
    DaoException;                                         //更新记录
  public void delete(Connection transConn,java.lang.String
    userName,java.lang.String roleName) throws DaoException;   //删除记录
  public PUserRoleVO findByPrimaryKey(Connection transConn,java.lang.
    String userName,java.lang.String roleName) throws DaoException;
                                                          //通过主键查询
  public java.util.Collection findAll(Connection transConn) throws
    DaoException;                                         //查询所有记录

  public java.util.Collection findByCondition(Connection
    transConn,java.util.Properties condition) throws DaoException;
                                                          //通过条件查询
  public java.util.Collection findByUser_Name(Connection transConn,
    java.lang.String userName ) throws DaoException;     //通过用户名查询
  public java.util.Collection findByRole_Name(Connection transConn,
    java.lang.String roleName ) throws DaoException;     //通过角色名查询
}
```

（3）DAO 组件的实现

本系统中 DAO 组件的实现类如表 8-10 所示。

表 8-10　DAO 组件的实现类表

DAO 组件的实现类	描　　述
PPopedomDAOImpl	权限 DAO 组件的实现
PRolePopedomDAOImpl	角色对应的权限 DAO 组件的实现
PUserDAOImpl	用户 DAO 组件的实现
PUserPopedomViewDAOImpl	用户权限视图 DAO 组件的实现
PUserRoleDAOImpl	用户对应的角色 DAO 组件的实现

下面以用户角色信息表为例进行设计说明。

PUserRoleDAOImpl（实现）类的代码设计如下：

```
package com.dfkj.pm.dao;
import java.sql.*;
import com.dfkj.pm.exception.*;
import com.dfkj.pm.data.*;
import com.dfkj.pm.vo.PUserRoleVO;
import java.util.*;

public class PUserRoleDAOImpl implements  IPUserRoleDAO
{
  public void insert(Connection transConn,PUserRoleVO pUserRoleVO) throws
         DaoException   //增加记录
  {
    Debug.println( this.getClass().getName() + " Insert begin");
    Connection conn = null;
    PreparedStatement statement = null;
    try
    {
       conn = transConn;
        //根据所选条件判断数据是否存在
       try
       {
          daoFindSame(conn,DaoUtil.NullToStr(pUserRoleVO.getUserName()),
          DaoUtil.NullToStr(pUserRoleVO.getRoleName()));
          throw new DuplicateDataException("Primary key already exists");
       }
       catch(ObjectNotFoundException objectNotFoundE)
       {
       }
       statement = conn.prepareStatement("INSERT INTO
          p_user_role(user_name , role_name) VALUES( ?  ,  ?  )");
       statement.setString(1, DaoUtil.NullToStr(pUserRoleVO.getUserName()));
       statement.setString(2, DaoUtil.NullToStr(pUserRoleVO.getRoleName()));
       if (statement.executeUpdate() != 1)
       {
          throw new  DaoException("Error adding row.");
       }
```

```
    }
    catch(SQLException e)
    {
        Debug.println(e.getMessage());
        throw new  DaoException("Error executing SQL INSERT INTO
                p_user_role(user_name , role_name) VALUES( ?  ,  ?  )"
                + e.getMessage());
    }
    finally
    {
        DBUtil.closeStatement(statement);
    }
    Debug.println( this.getClass().getName() + " Insert end");
  }
  public void update(Connection transConn,PUserRoleVO pUserRoleVO) throws
          DaoException   //修改记录
  {
    Debug.println( this.getClass().getName() + " Update begin");
    Connection conn = null;
    PreparedStatement statement = null;
    try
    {
        conn = transConn;
        statement = conn.prepareStatement(" UPDATE  p_user_role
          SET user_name = ?  , role_name = ?
          WHERE  user_name = ?  AND  role_name = ? ");
        statement.setString(1,DaoUtil.NullToStr(pUserRoleVO.getUserName()));
        statement.setString(2,DaoUtil.NullToStr(pUserRoleVO.getRoleName()));

        //条件
        statement.setString(3,DaoUtil.NullToStr(pUserRoleVO.getUserName()));
        statement.setString(4,DaoUtil.NullToStr(pUserRoleVO.getRoleName()));
        if (statement.executeUpdate() != 1)
        {
            throw new  ObjectNotFoundException("Error updating row");
        }
    }
    catch(SQLException e)
    {
        Debug.println(e.getMessage());
        throw new  DaoException("Error executing SQL  UPDATE  p_user_role
          SET user_name = ?  , role_name = ?
          WHERE  user_name = ?  AND  role_name = ? " + e.getMessage());
    }
    finally
    {
        DBUtil.closeStatement(statement);
    }
```

```
  Debug.println( this.getClass().getName() + " Update end");
 }
 public void delete(Connection transConn,java.lang.String
      userName,java.lang.String  roleName) throws DaoException
 {        //删除记录
  Debug.println( this.getClass().getName() + " Delete begin");
  Connection conn = null;
  PreparedStatement statement = null;
  try
  {
     conn = transConn;
     statement = conn.prepareStatement(" DELETE  p_user_role
       WHERE  user_name = ?  AND  role_name = ? ");
     //条件
     statement.setString(1, DaoUtil.NullToStr(userName));
     statement.setString(2, DaoUtil.NullToStr(roleName));
     if (statement.executeUpdate() != 1)
     {
        throw new  RemoveException("Error deleting row");
     }
  }
  catch(SQLException e)
  {
     Debug.println(e.getMessage());
     throw new  DaoException("Error executing SQL  DELETE  p_user_role
       WHERE  user_name = ?  AND  role_name = ? " + e.getMessage());
  }
  finally
  {
     DBUtil.closeStatement(statement);
  }
  Debug.println( this.getClass().getName() + " Delete end");
 }
 public PUserRoleVO findByPrimaryKey(Connection transConn,java.lang.String
   userName,java.lang.String  roleName) throws DaoException
 {   //通过主键查询记录
  Debug.println( this.getClass().getName() + " Select begin");
  Connection conn = null;
  PreparedStatement statement = null;
  ResultSet rs = null;
  PUserRoleVO pUserRoleVO = null;
  try
  {
     conn = transConn;
     statement = conn.prepareStatement(" SELECT user_name , role_name
     FROM  p_user_role WHERE  user_name = ?  AND  role_name = ? ");
     //条件
     statement.setString(1, DaoUtil.NullToStr(userName));
```

```
        statement.setString(2, DaoUtil.NullToStr(roleName));
        rs = statement.executeQuery();
        if (rs.next())
        {
          pUserRoleVO = new PUserRoleVO();
          pUserRoleVO.setUserName(rs.getString("user_name"));
          pUserRoleVO.setRoleName(rs.getString("role_name"));
        }
        else
        {
           throw new ObjectNotFoundException("Row does not exist.");
        }
      }
      catch(SQLException e)
      {
        Debug.println(e.getMessage());
        throw new  DaoException("Error executing SQL  SELECT user_name ,
          role_name FROM  p_user_role
          WHERE  user_name = ?  AND  role_name = ? " + e.getMessage());
      }
      finally
      {
        DBUtil.closeResultSet(rs);
        DBUtil.closeStatement(statement);
      }
      Debug.println( this.getClass().getName() + " Select end");
      return pUserRoleVO;
    }
   public java.util.Collection findAll(Connection transConn) throws
DaoException
   {    //查询所有记录
      Debug.println( this.getClass().getName() + " Select begin");
      Connection conn = null;
      PreparedStatement statement = null;
      ResultSet rs = null;
      Vector result = new Vector();
      PUserRoleVO pUserRoleVO = null;
      try
      {
        conn = transConn;
        statement = conn.prepareStatement(" SELECT user_name , role_name
                    FROM  p_user_role");
        rs = statement.executeQuery();
        while (rs.next())
        {
          pUserRoleVO = new PUserRoleVO();
          pUserRoleVO.setUserName(rs.getString("user_name"));
          pUserRoleVO.setRoleName(rs.getString("role_name"));
```

```
        result.add(pUserRoleVO);
      }
    }
    catch(SQLException e)
    {
      Debug.println(e.getMessage());
      throw new  DaoException("Error executing SQL  SELECT user_name ,
        role_name FROM  p_user_role" + e.getMessage());
    }
    finally
    {
      DBUtil.closeResultSet(rs);
      DBUtil.closeStatement(statement);
    }
    Debug.println( this.getClass().getName() + " Select end");
    return result ;
  }
  public java.util.Collection findByCondition(Connection
        transConn,java.util.Properties condition) throws DaoException
  {    //通过条件查询记录
    Debug.println( this.getClass().getName() + " Select begin");
    Connection conn = null;
    PreparedStatement statement = null;
    ResultSet rs = null;
    Vector result = new Vector();
    Vector fieldList = new Vector();
    String sql = "";
    PUserRoleVO pUserRoleVO = null;
    try
    {
      conn = transConn;
      sql = " SELECT user_name , role_name FROM  p_user_role";
      String whereClause = " WHERE 1=1 ";
      String fieldValue = null;
      //输出查询条件
      if (condition != null)
      {
        fieldValue = condition.getProperty("USER_NAME");
        if( fieldValue != null && fieldValue.length() > 0)
        {
          whereClause += " AND user_name = ?  ";
          fieldList.add(fieldValue);
        }
        fieldValue = condition.getProperty("ROLE_NAME");
        if( fieldValue != null && fieldValue.length() > 0)
        {
          whereClause += " AND role_name = ?  ";
          fieldList.add(fieldValue);
```

```
            }
        }
        sql += whereClause;
        //查询条件完毕
        statement = conn.prepareStatement(sql);
        for(int i=0; i<fieldList.size(); i++)
        {
          statement.setString(i+1,(String)fieldList.elementAt(i));
        }
        rs = statement.executeQuery();
        while (rs.next())
        {
          pUserRoleVO = new PUserRoleVO();
          pUserRoleVO.setUserName(rs.getString("user_name"));
          pUserRoleVO.setRoleName(rs.getString("role_name"));
          result.add(pUserRoleVO);
        }
    }
    catch(SQLException e)
    {
        Debug.println(e.getMessage());
        throw new  DaoException("Error executing SQL" + sql + e.getMessage());
    }
    finally
    {
        DBUtil.closeResultSet(rs);
        DBUtil.closeStatement(statement);
    }
    Debug.println( this.getClass().getName() + " Select end");
    return result ;
  }
  private void daoFindSame(Connection transConn, java.lang.String userName ,
      java.lang.String roleName ) throws DaoException
  {  //查找相同的数据
    Debug.println( this.getClass().getName() + " Select begin");
    Connection conn = null;
    PreparedStatement statement = null;
    ResultSet rs = null;
    Vector result = new Vector();
    PUserRoleVO pUserRoleVO = null;
    try
    {
        conn = transConn;
        statement = conn.prepareStatement(" SELECT user_name , role_name
          FROM  p_user_role
          WHERE 1=1  AND user_name = ?  AND role_name = ? ");
        //条件
        statement.setString(1, DaoUtil.NullToStr(userName));
```

```
            statement.setString(2, DaoUtil.NullToStr(roleName));
            rs = statement.executeQuery();
            if  (!rs.next())
            {
              //有相同的数据，抛出异常
              throw new ObjectNotFoundException("没有发现相同的数据！");
            }
        }
        catch(SQLException e)
        {
            Debug.println(e.getMessage());
            throw new  DaoException("Error executing SQL  SELECT user_name ,
              role_name FROM  p_user_role
              WHERE 1=1 AND user_name=? AND role_name=?" + e.getMessage());
        }
        finally
        {
            DBUtil.closeResultSet(rs);
            DBUtil.closeStatement(statement);
        }
        Debug.println( this.getClass().getName() + " Select end");
    }
    public java.util.Collection findByUser_Name(Connection transConn,
          java.lang.String userName ) throws DaoException
    {   //通过用户名查询记录
        Debug.println( this.getClass().getName() + " Select begin");
        Connection conn = null;
        PreparedStatement statement = null;
        ResultSet rs = null;
        Vector result = new Vector();
        PUserRoleVO pUserRoleVO = null;
        try
        {
            conn = transConn;
            statement = conn.prepareStatement(" SELECT user_name , role_name
              FROM  p_user_role WHERE 1=1  AND user_name = ? ");
            //条件
            statement.setString(1, DaoUtil.NullToStr(userName));
            rs = statement.executeQuery();
            while (rs.next())
            {
              pUserRoleVO = new PUserRoleVO();
              pUserRoleVO.setUserName(rs.getString("user_name"));
              pUserRoleVO.setRoleName(rs.getString("role_name"));
              result.add(pUserRoleVO);
            }
        }
        catch(SQLException e)
```

```
    {
      Debug.println(e.getMessage());
      throw new  DaoException("Error executing SQL  SELECT user_name ,
        role_name FROM  p_user_role
        WHERE 1=1  AND user_name = ? " + e.getMessage());
    }
    finally
    {
      DBUtil.closeResultSet(rs);
      DBUtil.closeStatement(statement);
    }
    Debug.println( this.getClass().getName() + " Select end");
    return result ;
  }
  public java.util.Collection findByRole_Name(Connection transConn,
        java.lang.String roleName ) throws DaoException //通过角色名称查询
  {
    Debug.println( this.getClass().getName() + " Select begin");
    Connection conn = null;
    PreparedStatement statement = null;
    ResultSet rs = null;
    Vector result = new Vector();
    PUserRoleVO pUserRoleVO = null;
    try
    {
      conn = transConn;
      statement = conn.prepareStatement(" SELECT user_name , role_name
         FROM  p_user_role WHERE 1=1  AND role_name = ? ");
      //条件
      statement.setString(1, DaoUtil.NullToStr(roleName));
      rs = statement.executeQuery();
      while (rs.next())
      {
        pUserRoleVO = new PUserRoleVO();
        pUserRoleVO.setUserName(rs.getString("user_name"));
        pUserRoleVO.setRoleName(rs.getString("role_name"));
        result.add(pUserRoleVO);
      }
    }
    catch(SQLException e)
    {
      Debug.println(e.getMessage());
      throw new  DaoException("Error executing SQL  SELECT user_name ,
        role_name FROM  p_user_role
        WHERE 1=1  AND role_name = ? " + e.getMessage());
    }
    finally
    {
      DBUtil.closeResultSet(rs);
```

```
        DBUtil.closeStatement(statement);
      }
      Debug.println( this.getClass().getName() + " Select end");
      return result ;
    }
}
```

（4）VO（数值对象）映射

本系统中 VO 映射类如表 8-11 所示。

表 8-11　VO 映射类表

VO 映射类	描　　述
PPopedomVO	权限 VO 映射
PRolePopedomVO	角色对应的权限 VO 映射
PUserPopedomViewVO	用户 VO 映射
PUserRoleVO	用户权限视图 VO 映射
PUserVO	用户对应的角色 VO 映射

下面以用户对应的角色信息表为例进行设计说明。

```
package com.dfkj.pm.vo;
public class PUserRoleVO implements java.io.Serializable
{
  public PUserRoleVO()
  {
  }
  private String userName = "";
  private String roleName = "";
  public void setUserName(String userName)      //设置用户名
  {
    this.userName = userName;
  }
  public String getUserName()                   //获取用户名
  {
    return userName;
  }
  public void setRoleName(String roleName)      //设置角色名称
  {
    this.roleName = roleName;
  }
  public String getRoleName()                   //获取角色名称
  {
    return roleName;
  }
}
```

❑　服务层设计

在本系统中采用 Struts 来构造系统的框架，因此这里所有的逻辑都是通过 Actions 和

Business 组合起来的，当然也可以将 Business 合并到 Actions。下面仅以信息汇报部分为例进行说明。

➢　actions 请求类

本系统中 Actions 请求类如表 8-12 所示。

表 8-12　Actions 请求类

Actions 请求类	描　　述
AddRoleAction	增加角色
AddRolePrepareAction	欲增加角色
AddUserAction	增加用户
AddUserPrepareAction	欲增加用户
AddUserRoleAction	给用户赋角色
DeleteRoleAction	删除角色
DeleteUserAction	删除用户
DeleteUserRoleAction	删除给用户赋的角色
ListAllUserAction	所有用户列表显示
ListRoleAction	角色列表显示
ListRolePopedomAction	显示角色对应权限列表
ListUserAction	用户列表显示
ListUserRoleAction	用户所含的角色列表显示
LoginAction	用户登录
LogoutAction	用户注销
ModifyRoleAction	修改角色
ModifyRolePrepareAction	欲修改角色
ModifyRoleUserAction	修改角色对应用户
ModifyRoleUserPrepareAction	欲修改角色对应用户
ModifyUserAction	修改用户
ModifyUserPassword	修改用户密码
ModifyUserPrepareAction	欲修改用户
ModifyUserRoleAction	修改用户对应角色
ModifyUserRolePrepareAction	欲修改用户对应角色

➢　Business 业务逻辑类

本系统中 Business 业务逻辑类设计如表 8-13 所示。

表 8-13　Business 业务逻辑类表

Business 业务逻辑类	描　　述
BusinessFactory	Business 工厂
ILogInfo	登录信息接口
IRole	角色接口
IUser	用户接口

续表

Business 业务逻辑类	描　　述
IUserRole	用户对应的角色接口
LogInfoImp	登录信息实现
RoleImp	角色实现
UserImp	用户实现
UserRoleImp	用户对应的角色实现

下面以 UserRoleImp（用户对应的角色实现）类的设计为例进行说明。

UserRoleImp 类的代码设计如下：

```
package com.dfkj.pm.business;
import java.util.*;
import java.sql.*;
import com.dfkj.pm.dao.*;
import com.dfkj.pm.vo.*;
import com.dfkj.pm.constants.*;
import com.dfkj.pm.exception.*;
public class UserRoleImp implements IUserRole { //用户对应的角色实现
    private IPUserRoleDAO iPUserRoleDAO;
    public UserRoleImp() {
        iPUserRoleDAO = DAOFactory.newInstance().buildPUserRoleDAO();
    }

    public void addUserRole(java.sql.Connection conn,  //增加用户对应的角色
      com.dfkj.pm.vo.PUserRoleVO pUserRole) throws DaoException {
            iPUserRoleDAO.insert(conn, pUserRole);
    }
    public void addUserRole(java.sql.Connection conn, java.util.Collection
    UserRoleCol)throws DaoException{  //批量增加用户对应的角色
        Vector vecUserRole = new Vector();
        PUserRoleVO pUserRoleVO = new PUserRoleVO();
        int i=0;
        vecUserRole = (Vector)UserRoleCol;
        try{
            conn.setAutoCommit(false);
            for(i=0;i<vecUserRole.size();i++){
                pUserRoleVO = (PUserRoleVO)vecUserRole.elementAt(i);
                addUserRole(conn,pUserRoleVO);
            }
            conn.commit();
            conn.setAutoCommit(true);
        }catch(SQLException e){
            try{
                conn.rollback();
            }catch(SQLException ex){
            }
            throw new
```

```
            DaoException(ExceptionConstants.SET_COMMIT_ERROR);
        }catch(DaoException ex2){
            try{
                    conn.rollback();
            }catch(SQLException ex){
            }
            throw ex2;
        }
    }
    public void deleteUserRole(java.sql.Connection conn, //删除用户对应的角色
     com.dfkj.pm.vo.PUserRoleVO pUserRole) throws DaoException {
        String UserName = null, RoleName = null;
        UserName = pUserRole.getUserName();
        RoleName = pUserRole.getRoleName();
        if(UserName.compareTo(Constants.ADMIN_USERNAME)==0){
            throw new
                DaoException(ExceptionConstants.CANNOT_DEL_ADMINROLE);
        }else{
            try{
                conn.setAutoCommit(false);
                iPUserRoleDAO.delete(conn, pUserRole.getUserName(),
                pUserRole.getRoleName());
                conn.commit();
            }catch(SQLException e){
                try{
                    conn.rollback();
                }catch(SQLException ex){
                }
                throw new
                DaoException(ExceptionConstants.SET_COMMIT_ERROR);
            }catch(DaoException ex1){
                try{
                    conn.rollback();
                }catch(SQLException ex){
                }
                throw ex1;
            }
        }
    }
    public void deleteUserRole(java.sql.Connection conn, java.util.Collection
      UserRoleCol) throws DaoException    //批量删除用户对应的角色
    {
        Vector vecUserRole = new Vector();
        int i=0;
        vecUserRole = (Vector)UserRoleCol;
        try{
                conn.setAutoCommit(false);
                for(i=0;i<vecUserRole.size();i++){
```

```
                deleteUserRole(conn, (PUserRoleVO)vecUserRole.elementAt(i));
            }
            conn.commit();
        }catch(SQLException e){
            try{
                conn.rollback();
            }catch(SQLException ex){
            }
            throw new
              DaoException(ExceptionConstants.SET_COMMIT_ERROR);
        }catch(DaoException ex2){
            try{
                conn.rollback();
            }catch(SQLException ex){
            }
            throw ex2;
        }
    }
    public java.util.Collection findUserRoleByRole(java.sql.Connection conn,
String
    RoleName) throws DaoException { //通过角色名称查询用户对应的角色
        Vector vecUserRole = new Vector();
        vecUserRole = (Vector)iPUserRoleDAO.findByRole_Name(conn,
          RoleName);
        return vecUserRole;
    }
   public java.util.Collection findUserRoleByUser(java.sql.Connection conn,
String
    UserName) throws DaoException {  //通过用户名称查询
        Vector vecUserRole = new Vector();
        vecUserRole = (Vector)iPUserRoleDAO.findByUser_Name(conn,
        UserName);
        return vecUserRole;
    }
}
```

8.5　运行与调试本章的案例

将源代码中提供的本章应用程序（整个目录 USER）复制到本机所安装的\Apache Software Foundation\Tomcat 5.5\webapps 路径下。

由于本章提供的案例是在 Oracle 8.17 数据库下开发的，所以读者可以在 Oracle 数据库下创建用户 eoa，其密码为 eoa，然后将 eoa.dmp 直接导入到数据库即可。

另外，读者可以将程序进行简单修改（主要是 DAO），将数据库迁移到 MySQL 5.1。

在 Tomcat 5.5 中要进行调试，跟踪调试信息，可以直接运行 Tomcat 5.5 中的 tomcat5.exe 程序。

（1）配置环境变量

在"我的电脑"上单击鼠标右键，在弹出的快捷菜单中选择"属性"命令，弹出"系统特性"对话框，选择"高级"选项卡，然后单击"环境变量"按钮，即可编辑系统的环境变量。

（2）设置 Server.xml 文件

Tomcat 主目录/conf 下的 server.xml 文件是对 Web 服务器的配置。找到以下代码：

```
<Connector
       URIEncoding="GBK"
       port="8080"
       redirectPort="8443"
       minSpareThreads="25"
       connectionTimeout="20000"
       uRIEncoding="GBK"
       maxSpareThreads="75"
       maxThreads="150"
       maxHttpHeaderSize="8192">
</Connector>
```

可以将 8080 端口改为喜欢使用的端口，如常见的 80（只要不冲突），以后即可利用该端口访问系统 http://localhost:80/USER。

如果本章案例应用程序（\USER 整个目录）不放在\Apache Software Foundation\Tomcat 5.5\webapps 路径下，例如放在开发路径 d:\myeclipseproject 下，那么在 server.xml 文件中找到以下代码：

```
<Host
      appBase="webapps"
      name="localhost">
</Host>
```

在其间添加一个<Context>元素：

```
<Context path="/tyxy"  reloadable="true"
   docBase=" d:\myeclipseproject\USER"
   workDir=" d:\myeclipseproject\USER\work" >
</Context>
```

在浏览器中打开 http://localhost:80/USER 时就会转向 d:\myeclipseproject\USER 下的应用程序。

（3）设置 web.xml 文件

刚开始调试时，可以对\Tomcat 5.5\webapps\USER\WEB-INF 下的 web.xml 中的代码进行修改，如下所示：

```
<filter>
   <!-- 权限过滤器定义 -->
       <filter-name>com.dfkj.web.PowerFilter</filter-name>
       <filter-class>com.dfkj.web.PowerFilter</filter-class>
       <init-param>
```

```
            <param-name>flag</param-name>
            <param-value>off</param-value>  <!--off 表示权限有效，on 无效 -->
        </init-param>
    </filter>
```

将 off 改为 on，这样权限控制将不起作用，输入用户名和密码，可以进入任何模块，主要是为了调试方便。

8.6 小　　结

本章详细地讲述了利用 J2EE 架构构建通用用户权限管理系统的过程。通过本章的学习，读者对 Java B/S 结构的软件开发有了一定的认识。读者可以在本系统源代码基础上进行修改，以更好地符合实际项目需求。

第 9 章　CASE 支撑系统案例

本章通过一个软件项目“CASE 支撑系统”的设计与开发，实现在网上进行故障的申告、跟踪、处理等基本功能。描述了在 Web 中间件 Tomcat 环境下，采用 J2EE 技术和 B/S 结构，如何一步一步实现用户所要求的功能和性能。

9.1　系统需求分析

随着信息化程度的提高，很多单位成立了信息中心对信息系统的故障和咨询信息进行处理。有些单位信息系统众多，系统结构各不相同，应用平台也不统一，硬件网络等设施结构复杂，需要不同的专门人员去维护不同的分系统，要求一套 CASE 系统跟踪不同的处理需求，及时了解不同系统故障的处理进度，进行联合指导协调跨专业系统的故障。通过历史数据的沉淀，规范系统支撑的流程体系，积累系统故障处理经验，形成系统处理的知识库，更好地为日常维护工作提供支撑。

9.1.1　需求概述

CASE 系统是一套基于计算机的故障处理跟踪系统，能对故障 CASE 从受理创建到关闭的所有环节进行跟踪，自动记录创建及关闭日期，系统自动记录 CASE 处理状态的时间，在规定的期限内会按照优先级及系统的不同，向该系统故障负责人的上级主管发送告警信息（短信、邮件），在规定的时间没有处理完成的故障，系统会自动重复地告警，直到 CASE 关闭或故障的级别变得更低，不需要自动告警。系统在 CASE 受理后，根据配置的区别，会向呼叫者发送短信通知，在故障处理完成后，系统会提示用户故障处理完成。系统自动记录每个环节的处理时间，可以分析处理该故障的平均时间，有利于对 CASE 处理人员进行考核。

根据上面的大概需求，画出 CASE 支撑系统的用例图，如图 9-1 所示。

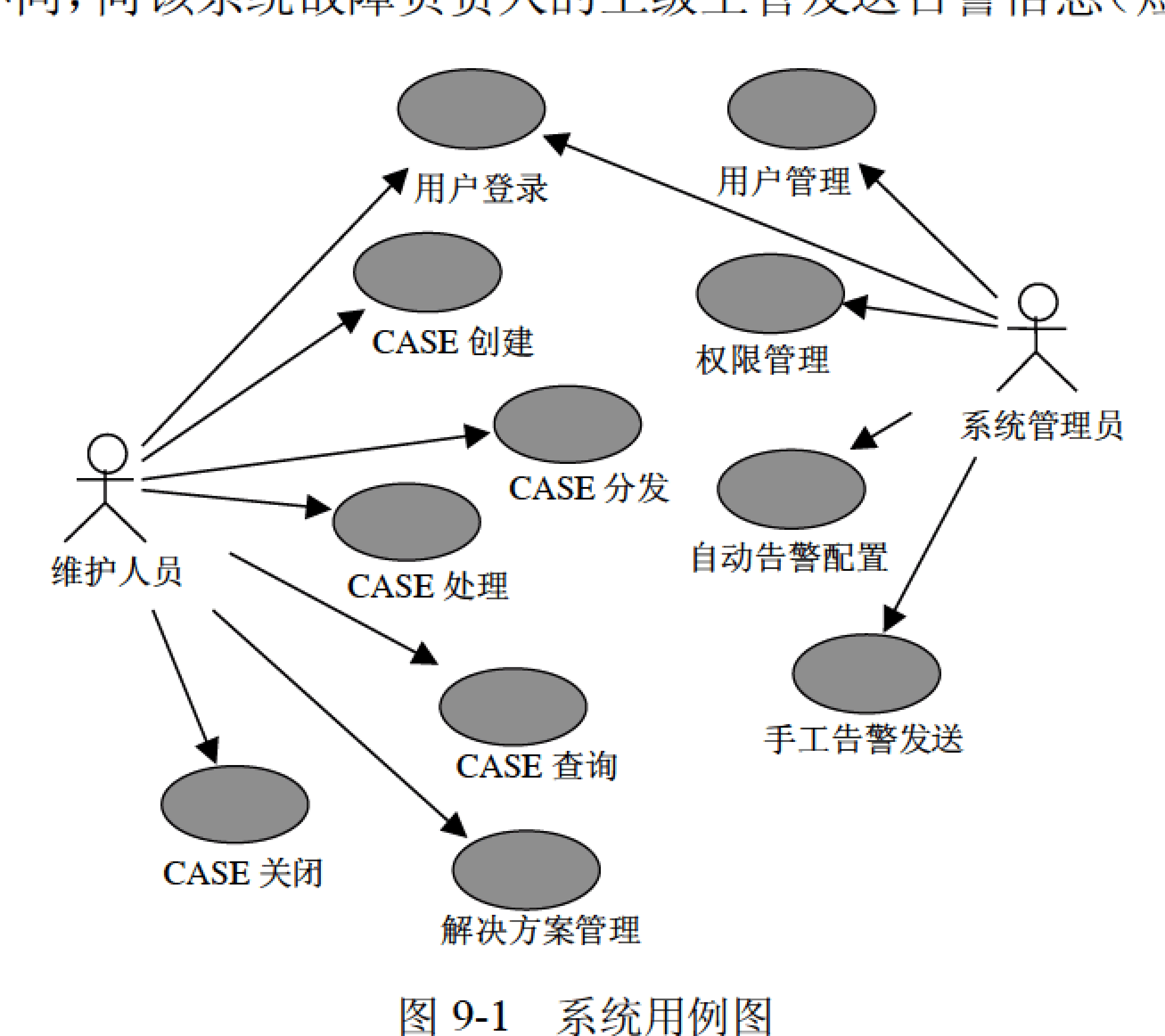


图 9-1　系统用例图

9.1.2　系统功能描述

通过前面的初步需求分析，下面来描述每个模块功能。

1．用户登录

在图 9-2 所示的登录页面中输入用户名和密码，然后单击“登录”按钮提交进行登录。各用户均须提供有效的用户名和密码才能进入 CASE 支撑系统。如果登录失败，系统会出现提示信息。



图 9-2　登录界面

2．CASE 创建

（1）单击“CASE 管理”菜单下的到 CASE 创建项，弹出创建 CASE 页面，如图 9-3 所示。



图 9-3　CASE 创建页面

（2）在图 9-3 所示的页面中，选择发起 CASE 的呼叫者即 CASE 的提出者、该 CASE 的分类、CASE 所在的系统类别即该 CASE 所对应的系统和 CASE 的紧急级别，输入 CASE 内容，输入完毕单击“确定”按钮进行保存。

（3）CASE 创建成功后，到 CASE 分发页面进行分发操作。

（4）当出现类似存在的 CASE 可单击“解决方案查询”按钮，在解决方案信息列表中

查询 CASE 的解决方法，进行当场解决无须走 CASE 流程。

3. CASE 分发

（1）单击“CASE 管理”菜单下的到 CASE 分发项，弹出分发 CASE 查询页面，如图 9-4 所示。

（2）在图 9-4 所示的页面中，选择要进行分发的 CASE，单击操作列表项下的“分发”按钮，弹出分发页面，如图 9-5 所示。



图 9-4　CASE 分发查询页面



图 9-5　CASE 分发页面

（3）在图 9-5 所示的页面中输入分发意见，及处理该 CASE 的接收人，接收人可以进行多选，选择完毕单击“确定”按钮进行分发操作。

（4）分发成功后，当接收人登录系统就会在首页面上显示相关的任务列表。

4. CASE 处理

（1）单击“CASE 管理”菜单下的到 CASE 处理项，弹出处理 CASE 页面，如图 9-6 所示。

（2）在图 9-6 所示的页面中，选择要进行处理的 CASE，单击操作列表项下的“处理”按钮，弹出处理页面，如图 9-7 所示。在页面中输入处理内容及处理后的状态，选择完毕后单击“确定”按钮进行保存操作。

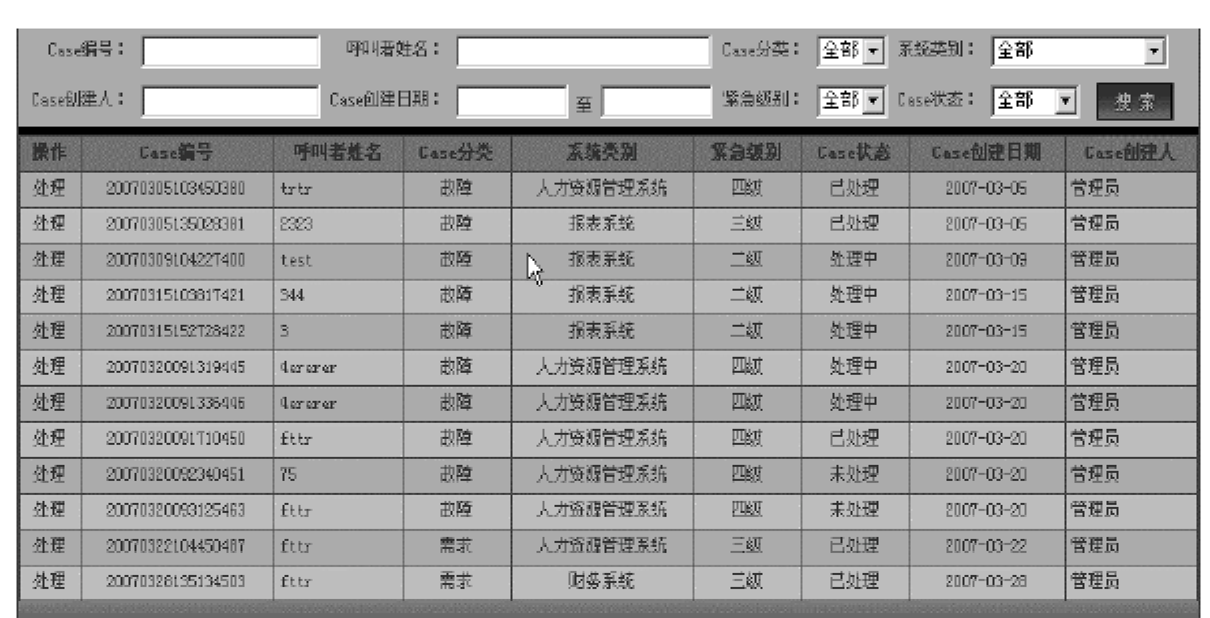



图 9-6　CASE 处理查询页面

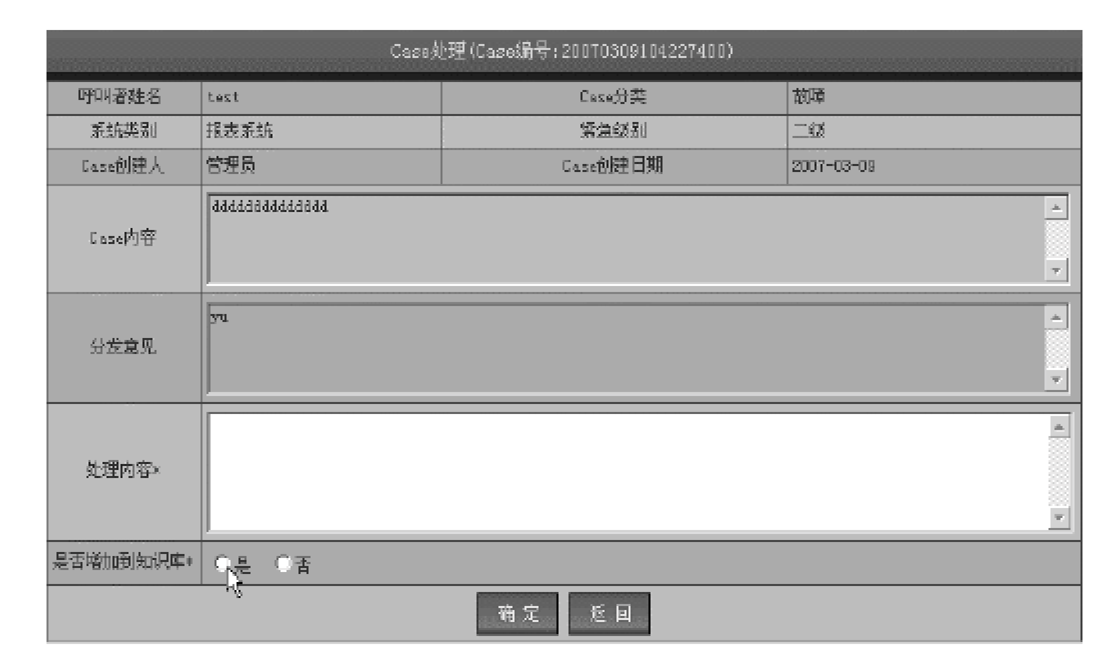


图 9-7　CASE 处理页面

（3）当该条 CASE 分发到多人时，有一人进行处理后，其他人就不需要进行处理操作。

（4）当该条 CASE 需要增加到知识库时选中“是”单选按钮，该 CASE 就插入“解决方案管理”的数据库表中，当下次遇到类似的 CASE 就知道处理方法了，否则，选中“否”

单选按钮。

（5）当 CASE 处理成功后，系统会给呼叫者发送处理成功的告警信息，以邮件或者短信方式进行告警。表明该条 CASE 已处理完毕。

5. CASE 关闭

（1）单击“CASE 管理”菜单下的到 CASE 关闭项，弹出关闭 CASE 页面，如图 9-8 所示。

（2）在图 9-8 所示的页面中，选择要进行关闭的 CASE，单击操作列表项下的“关闭”按钮，弹出关闭 CASE 页面，如图 9-9 所示。在页面中输入关闭意见，输入完毕后单击“确定”按钮进行保存操作。

图 9-8　CASE 关闭查询页面

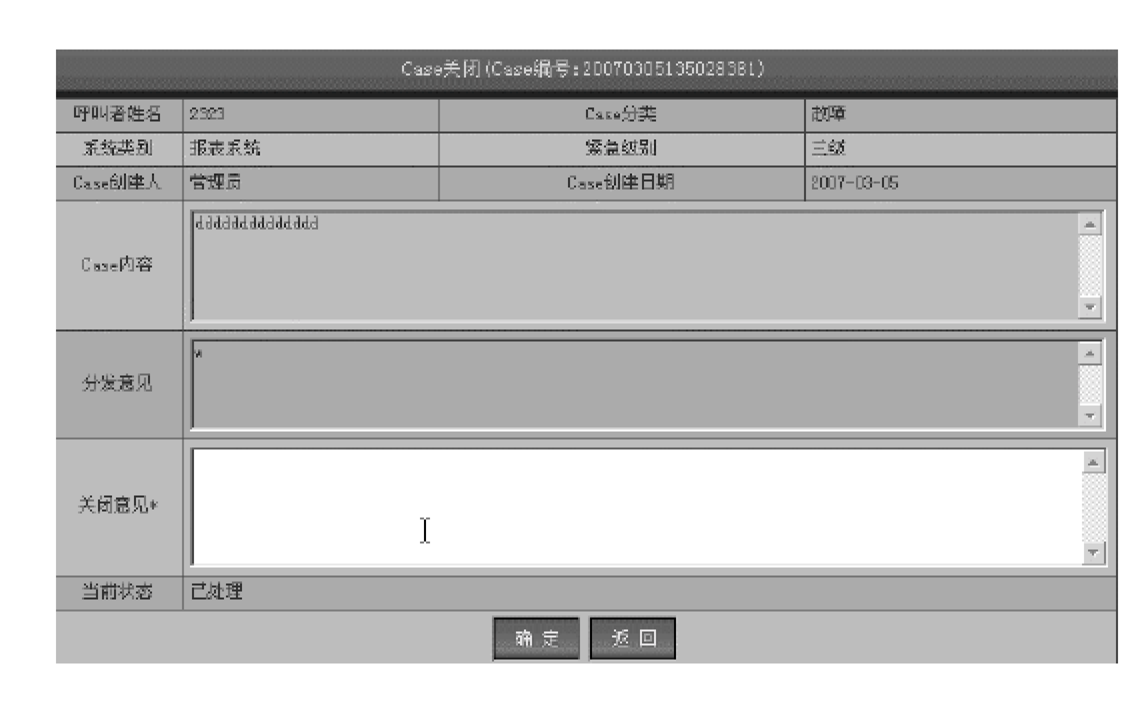


图 9-9　CASE 关闭页面

（3）CASE 关闭后，系统会向呼叫者以邮件或短信的方式发送告警信息，表明该条 CASE 已关闭。

6. 呼叫者管理

增加呼叫者信息：单击该页面中的增加按钮，在弹出的增加页面中录入相应的信息，录入信息页面如图 9-10 所示。输入带*标记的信息，输入完毕后单击“确定”按钮进行保存。



图 9-10　增加呼叫者页面

7. 自动告警配置

在此模块中设置好告警的接收人，当 CASE 创建成功后就会发邮件到接收人邮箱，通知接收人进行处理。

增加自动告警配置如图 9-11 所示。

图 9-11　自动告警增加页面

确诊时间：表示 CASE 创建时间到告警的时间段，如故障等级为一级故障的确诊时间为 1~3，创建的 CASE 也是一级故障，在上午 8 点创建的，如果在 9 点之前该条 CASE 还没有关闭，那么 9 点～11 点之间，系统都会间断地向接收人发送邮件通知（间隔时间在程序中根据实际情况而定），直到该条 CASE 关闭，若在确诊时间内该条 CASE 还未关闭，那么系统会向更高级别的接收人发送邮件进行告警，这也要在系统中事先配置好。

8. 手动告警发送

在此模块中直接选择未关闭的 CASE 编号发邮件到接收人，通知接收人进行处理。

手工告警发送如图 9-12 所示。

图 9-12　手工告警发送页面

在图 9-12 所示的页面中，在“未关闭 Case 编号”文本框后面单击“选择 Case 编号”按钮，在弹出的窗口中选择需要进行手工告警的 CASE，输入告警标题、告警内容、邮件告警、短信告警，输入完毕后，单击“发送”按钮，当发送成功后系统返回成功的信息，否则返回失败的信息。

当有多个接收人时，输入的 E-mail 地址和手机或小灵通用“;”隔开。

手工告警发送的 CASE 可通过 CASE 查询模块中进行查看，在 CASE 管理菜单下单击 CASE 查询，选中进行手工告警发送的 CASE，再单击“历史”按钮，在打开的窗口中可以看到手工告警发送的信息，如图 9-13 所示。

Case操作历史(Case编号:20070402170248530)　返 回

	操作时间	操作人	操作内容	状态变化
1	2007-04-02 17:02:48	clf	Case创建成功。	新创建（未分发）
2	2007-04-02 17:02:58	clf	Case分发给[clf]成功。	未分发->已分发（未接收）
3	2007-04-02 17:03:03	clf	Case接收成功。	未接收->已接收（处理中）
4	2007-04-02 17:03:07	clf	处理内容:test	处理中->已处理
5	2007-04-02 17:03:07	clf	手工编号为[20070402170248530]的Case告警到 cubef@hotnail.con 成功。	
6	2007-04-02 17:03:07	clf	手工短信发送编号为[20070402170248530]的Case告警到 13575766234 成功。	
7	2007-04-02 17:07:36	clf	处理内容:test	已处理->关闭
8	2007-04-02 17:07:36	clf	删除分发列表成功。	
9	2007-04-02 17:07:38	clf	手工编号为[20070402170248530]的Case告警到 chenlifang061102@126.com 成功。	
10	2007-04-02 17:07:38	clf	手工短信发送编号为[20070402170248530]的Case告警到 13575766234 成功。	

返 回

图 9-13　手工告警发送的历史信息页面

9．解决方案管理

- 增加解决方案：单击该页面中的增加按钮，在弹出的增加页面中录入相应的信息，输入完毕后单击“确定”按钮进行保存。
- 修改解决方案：选定需要修改题的记录，单击“操作”列表项下的“修改”按钮，在弹出的页面中输入相关的修改信息，单击“确定”按钮进行保存。
- 删除解决方案：选定需要删除的解决方案，单击“操作”列表项下的“删除”按钮进行删除操作。

该页面显示的解决方案信息有直接新增的，或者是从 CASE 处理页面中增加进来的。

9.2　系统总体架构

用例图分析了该应用程序的主要功能需求，这些需求是设计开发的依据。下面从宏观上开始讲解整个系统应用的设计要点。整个应用程序遵循多层次的架构模式。从上到下依次为视图层、控制层、模型层、JDBC 层和数据库层，如图 9-14 所示。

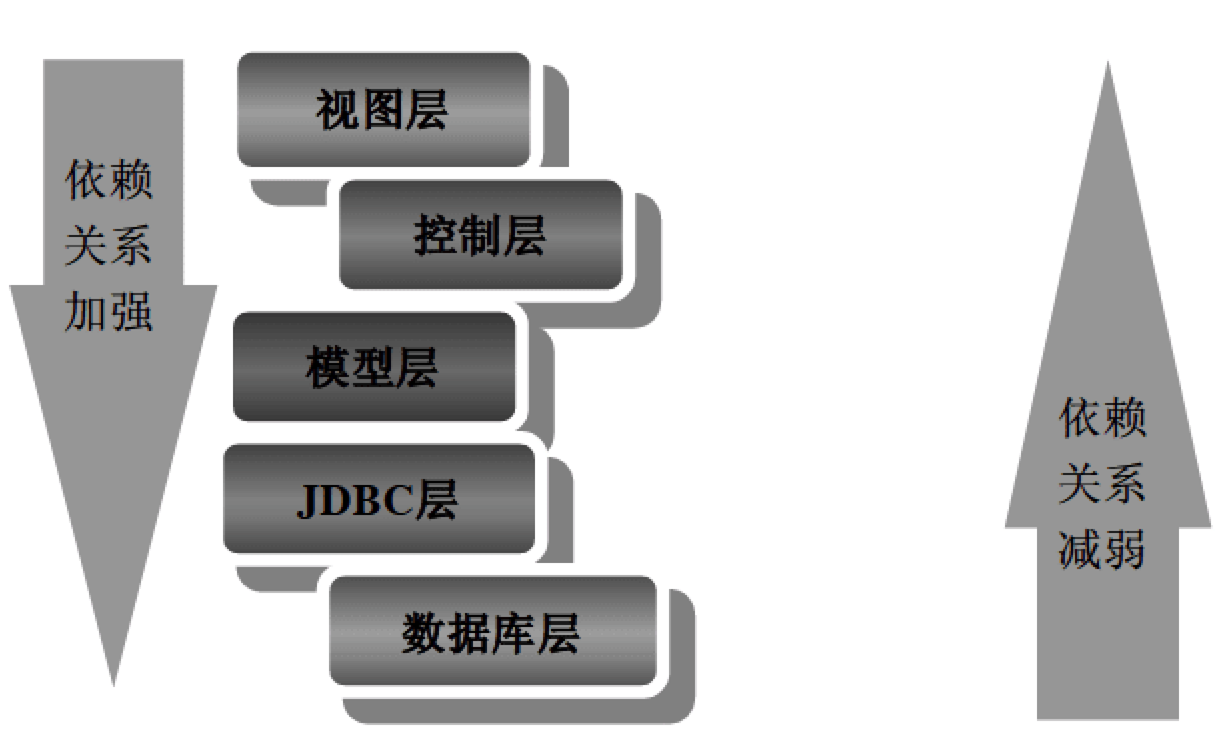


图 9-14　层次架构

系统总体框架如图 9-15 所示。

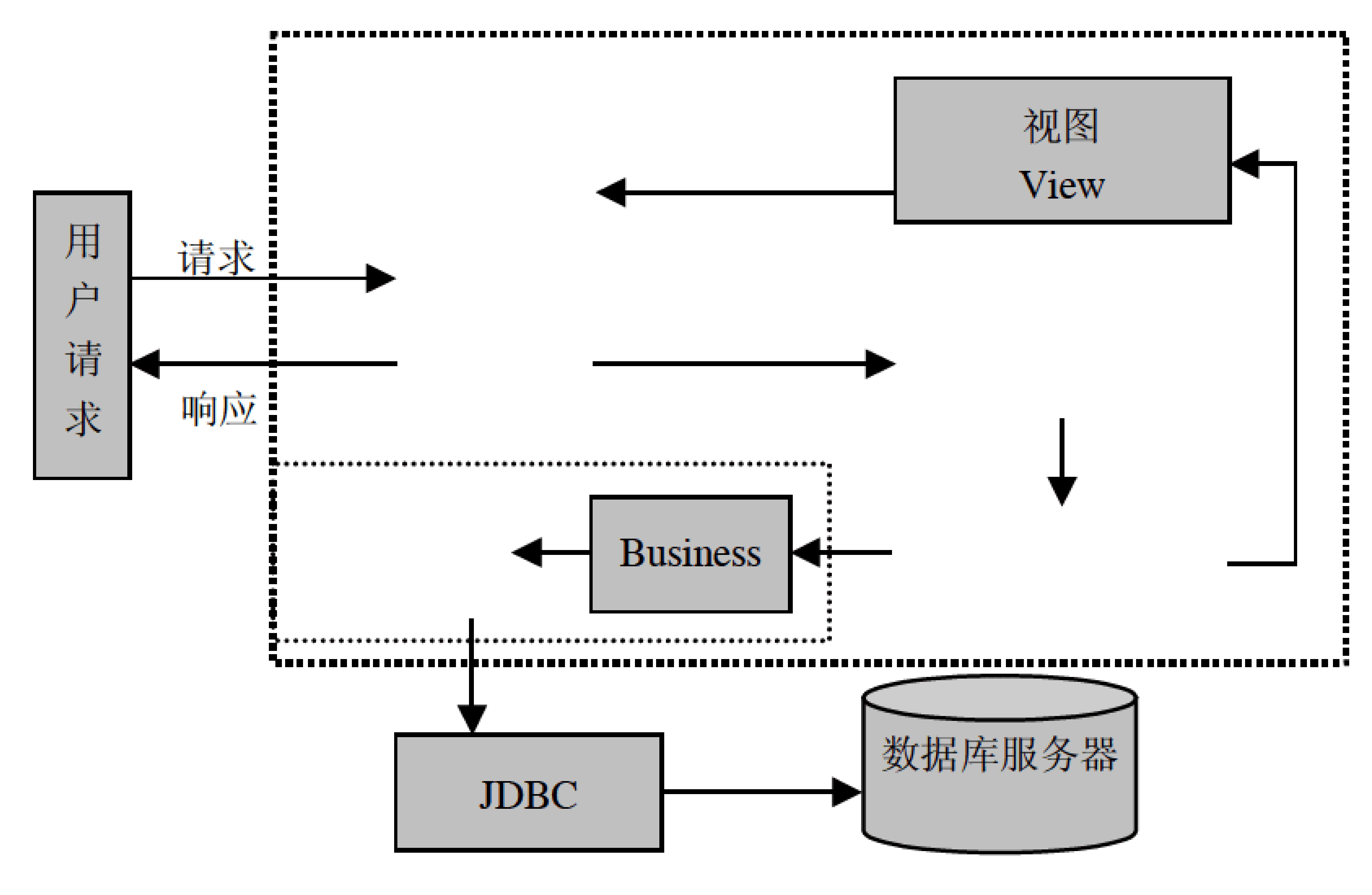


图 9-15 系统总体框架

9.3 数据库设计

数据库设计需要包括业务实体设计和数据模型设计。

9.3.1 业务实体设计

一个系统的业务实体在计算机内存中表现为实体域对象，在数据库中表现为关系数据，实现业务实体包括以下内容：

- 设计域模型，创建域模型实体对象。
- 设计关系数据模型。
- 创建对象—关系映射文件。

根据前面的系统需求分析，本系统中可以抽象出来的业务实体包括 CASE 处理活动、CASE 信息、告警队列、告警上报配置信息、呼叫者、CASE 解决方案等。这几个实体相互进行交互，下面介绍这些实体模型的定义。

- CASE 处理活动。代表一个 CASE 处理活动实体，主要属性有处理人、处理内容、状态变化、处理时间等。
- CASE 信息。代表一个 CASE 信息实体，主要属性包括呼叫者、CASE 分类、系统类别、紧急级别、CASE 内容、CASE 状态、分发意见、创建人、创建日期、关闭日期等。
- 告警队列。代表一个告警队列实体，主要属性有 CASE 流水号、下次告警时间等。
- 告警上报配置信息。代表一个告警上报配置信息实体，主要属性有 CASE 分类、系统类别、紧急等级、确诊时间、接收人姓名、接收人手机、接收人 E-mail 等。
- 呼叫者。代表一个呼叫者实体，主要属性包括呼叫者姓名、呼叫者公司、呼叫者

部门、电话号码、E-mail 等。

- ❑ CASE 解决方案。代表一个 CASE 解决方案实体，主要属性有系统类别、标题、问题描述、解决方案等。

9.3.2　数据模型设计

根据以上的关系来构造系统的数据模型。关系数据模型如图 9-16 所示。

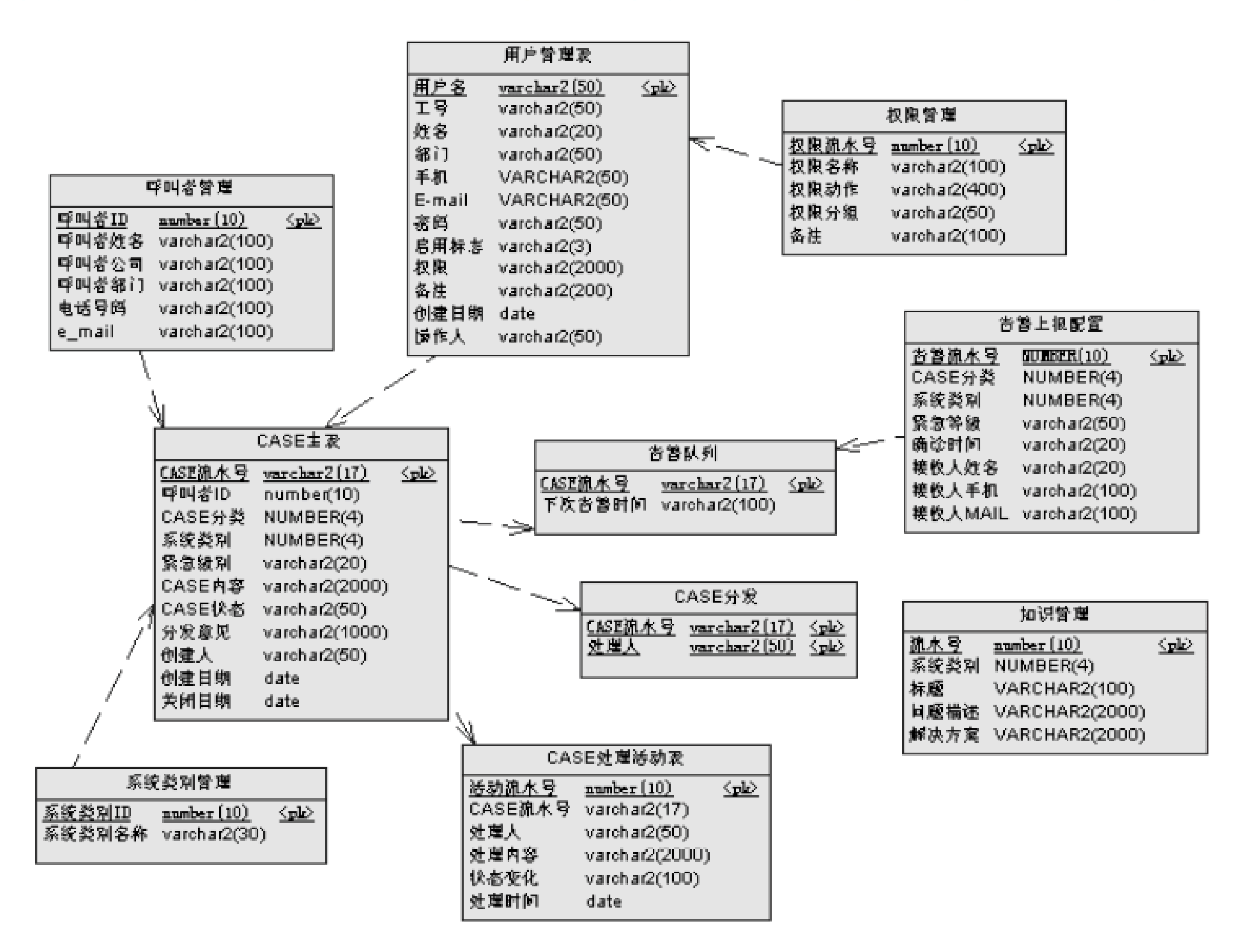


图 9-16　业务实体关系

下面建立各个数据模型。具体设计情况如下各表所示。

1．CASE 处理活动表（cc_case_hd）

表 9-1 主要用来保存 CASE 处理活动的有关信息，包括处理人、处理内容、状态变化、处理时间等。

表 9-1　CASE 处理活动表

字　段　名	中 文 含 义	字 段 类 型	长　　度	备　　注
hd_id	活动流水号	INT	10	主键
case_id	CASE 流水号	VARCHAR	17	
cl_ren	处理人	VARCHAR	50	
cl_nr	处理内容	VARCHAR	2000	
zt_bh	状态变化	VARCHAR	100	状态变化记录方法：前状态->后状态
cl_date	处理时间	DATE		

2．CASE 分发表（cc_ff）

CASE 分发表如表 9-2 所示。

表 9-2　CASE 分发表

字 段 名	中 文 含 义	字 段 类 型	长 度	备 注
case_id	CASE 流水号	VARCHAR	17	主键
cl_ren	处理人	VARCHAR	50	主键

3. CASE 信息表（cc_case）

表 9-3 主要用来保存 CASE 信息，包括审批时间、内容、审批人的信息等。

表 9-3　CASE 信息表

字 段 名	中 文 含 义	字 段 类 型	长 度	备 注
case_id	CASE 流水号	VARCHAR	17	主键；格式：yyyy+mm+dd+hh24+mi+ss+seq_case
hj_id	呼叫者 ID	INT	10	
cc_type	CASE 分类	INT	4	1-需求；2-故障
type	系统类别	INT	4	
gz_jb	紧急级别	VARCHAR	20	p1.一级；p2.二级；p3.三级；p4.四级
gz_nr	CASE 内容	VARCHAR	2000	
case_zt	CASE 状态	VARCHAR	50	状态值：Open；Send；Receive；Deal；Close
ff_yj	分发意见	VARCHAR	1000	
case_cjr	创建人	VARCHAR	50	
cj_date	创建日期	DATE		
close_date	关闭日期	DATE		

4. 告警队列表（cc_gxdl）

表 9-4 主要用来保存告警队列的有关信息。

表 9-4　告警队列表

字 段 名	中 文 含 义	字 段 类 型	长 度	备 注
case_id	CASE 流水号	VARCHAR	17	主键
next_time	下次告警时间	VARCHAR	100	格式为：开始时间-结束时间

5. 告警上报配置表（cc_gxsbpz）

表 9-5 主要用来保存告警上报配置有关信息。

表 9-5　告警上报配置表

字 段 名	中 文 含 义	字 段 类 型	长 度	备 注
gj_id	告警流水号	INT	10	主键
cc_type	CASE 分类	INT	4	
gj_type	系统类别	INT	4	

续表

字 段 名	中 文 含 义	字 段 类 型	长　度	备　注
gj_dj	紧急等级	VARCHAR	50	
gj_sj	确诊时间	VARCHAR	20	格式：0~1、1~4 等，判断方法：≥0 且<1，前者必须小于后者（单位：小时）
gj_jsrxm	接收人姓名	VARCHAR	20	
gj_phone	接收人手机	VARCHAR	100	
gj_mail	接收人 MAIL	VARCHAR	100	

6．呼叫者信息表（cc_hj）

表 9-6 主要用来保存呼叫者有关信息。

表 9-6　呼叫者信息表

字 段 名	中 文 含 义	字 段 类 型	长　度	备　注
hj_id	呼叫者 ID	INT	10	主键
hj_xm	呼叫者姓名	VARCHAR	100	
hj_gs	呼叫者公司	VARCHAR	100	
hj_dept	呼叫者部门	VARCHAR	100	
hj_dh	电话号码	VARCHAR	100	
e_mail	E-mail	VARCHAR	100	

7．CASE 解决方案表（cc_knowledge）

表 9-7 主要用来保存 CASE 解决方案的有关信息。

表 9-7　CASE 解决方案表

字 段 名	中 文 含 义	字 段 类 型	长　度	备　注
kl_id	流水号	INT	10	主键
kl_type	系统类别	INT	4	
kl_title	标题	VARCHAR	100	
gz_ms	问题描述	VARCHAR	2000	
jj_fa	解决方案	VARCHAR	2000	

9.4　系统详细设计

系统详细设计包括界面设计、逻辑主线、系统视图设计、系统包设计、数据库访问连接设计和业务层设计。

9.4.1　界面设计

在实际软件项目开发中，界面设计是需求分析之后首先要做的一件事，通过设计的界

面来和用户进行沟通，从而达成一致的需求。

在本系统的界面设计中，使用 CSS 进行界面设计。CSS 文件在本书所提供的源代码中，在工程文件夹\CASE\css 中有一个 CSS 文件 case.css 和\CASE\images 中有多个 GIF 文件、JPG 文件。

9.4.2　逻辑主线

一个 Java Web 应用所需要的一个核心文件，就是 web.xml。该文件控制整个应用的行为方式和方法，这是通过各种命令参数的配置来实现的，这些参数在服务启动时自动加载。web.xml 中的元素和 Tomcat 容器完全独立，它是 Web 应用的配置文件。

这里主要介绍贯穿整个系统的 Web 服务基本配置文件 Web.xml，其位于 CASE\WEB-INF 下。

- web.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>   <!-- 指定版本和文字编码 -->
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd"><!-- 指定 DTD 文档的位置
-->
<web-app>
  <display-name>CASE 支撑系统</display-name>
    <!-- 标记特定的 Web 应用的名称 -->
  <filter>
    <!--过滤器定义。将一个名字与一个实现 Javax.servlet.Filter 接口类关联。
    这里定义一个过滤器，名为 MainFilter。实现这个过滤器的类是
    com.dfkj.web.MainFilter -->
    <filter-name>com.dfkj.web.MainFilter</filter-name>
    <filter-class>com.dfkj.web.MainFilter</filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>GBK</param-value>
    </init-param>
    <init-param>
      <param-name>debug_flag</param-name>
      <param-value>on</param-value>
    </init-param>
  </filter>
  <filter>  <!-- 权限过滤器定义 -->
    <filter-name>com.dfkj.web.PowerFilter</filter-name>
    <filter-class>com.dfkj.web.PowerFilter</filter-class>
    <init-param>
      <param-name>flag</param-name>
      <param-value>on</param-value>  <!--off 表示权限有效，on 无效 -->
    </init-param>
  </filter>
  <filter-mapping>
<!—-主过滤器映射：一旦命名了一个过滤器，就要利用 filter-mapping 元素把它
```

```
与一个或多个 Servlet 或 JSP 页面相关联 -->
    <filter-name>com.dfkj.web.MainFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <filter-mapping>   <!--权限过滤器映射-->
    <filter-name>com.dfkj.web.PowerFilter</filter-name>
    <url-pattern>/MainController.do</url-pattern>
  </filter-mapping>
  <servlet>
        <!--Servlet 定义：在向 Servlet 或 JSP 页面制定初始化参数或定制 URL 时，
        必须首先命名 Servlet 或 JSP 页面。Servlet 元素就是用来完成此项任务的。
        这里定义了一个名为 MainController 的 Servlet ，实现这个 Servlet 的类是
        com.dfkj.web.MainController -->
    <servlet-name>MainController</servlet-name>
    <display-name>MainController</display-name>
    <description>用于改变字符集 </description>
    <servlet-class>com.dfkj.web.MainController</servlet-class>
    <init-param>
      <param-name>Debug</param-name>
      <param-value>on</param-value>
    </init-param>
  </servlet>
  <servlet>
    <servlet-name>ChartServlet</servlet-name>
    <servlet-class>com.dfkj.ec.chart.ChartServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>Log4jInit</servlet-name>
    <display-name>Log4jInit</display-name>
    <description>Log4j 初始化</description>
    <servlet-class>com.dfkj.log.Log4JInit</servlet-class>
    <init-param>
      <param-name>Log4jInitFile</param-name>
      <param-value>WEB-INF/classes/Log4j.properties</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>DbInitConfig</servlet-name>
    <servlet-class>com.dfkj.db.DbInitConfig</servlet-class>
    <init-param>
      <param-name>DbInitFile</param-name>
      <param-value>WEB-INF/classes/adapter.xml</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>DBServ</servlet-name>
    <display-name>DBServ</display-name>
```

```
    <description>用于初始化数据库连接</description>
    <servlet-class>com.dfkj.db.DBServ</servlet-class>
    <load-on-startup>4</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>validate</param-name>
      <param-value>true</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>treebuilders</param-name>
     <param-value>org.apache.webapp.admin.DepartTreeBuilder</param-value>
    </init-param>
    <init-param>
      <param-name>application</param-name>
      <param-value>ApplicationResources</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>AutoAlert</servlet-name>
    <servlet-class>com.dfkj.cc.actions.AutoAlert</servlet-class>
    <load-on-startup>5</load-on-startup>
  </servlet>
  <servlet-mapping>
<!-- Servlet 映射：服务器一般为 Servlet 提供一个默认的
    URL:http://host/webAppPrefix/servlet/ServletName。但常常会更改这个
    URL，以便 Servlet 可以访问初始化参数或更容易地处理相对 URL 时，
    使用 servlet-mapping 元素-->
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Log4jInit</servlet-name>
    <url-pattern>/servlet/Log4jInit</url-pattern>
```

```
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>MainController</servlet-name>
    <url-pattern>/MainController.do</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ChartServlet</servlet-name>
    <url-pattern>/note/ChartServlet.chart</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>DbInitConfig</servlet-name>
    <url-pattern>/servlet/com.dfkj.db.DbInitConfig</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>DBServ</servlet-name>
    <url-pattern>/servlet/com.dfkj.db.DBServ</url-pattern>
  </servlet-mapping>
  <error-page>    <!-- 错误处理页：使得在返回特定 HTTP 状态代码时，或者特定
类型的异常被抛出时，能够指向将要显示的错误处理页面-->
    <error-code>404</error-code>
    <location>/notFileFound.jsp</location>
  </error-page>
  <servlet-mapping>
    <servlet-name>AutoAlert</servlet-name>
    <url-pattern>/AutoAlert</url-pattern>
  </servlet-mapping>
  <session-config>   <!-- 控制会话超时。这里 Session 可以保持不活动状态的最长
                     时间为 360 秒，超过这一时间，Servlet 容器将把它作为无效
                     Session 处理-->
    <session-timeout>
          360
    </session-timeout>
  </session-config>
  <mime-mapping>  <!—MIME 类型映射：如果 Web 应用具有特殊的文件，希望能
    保证给它们分配特定的 MIME 类型，则 mime-mapping 元素提供这种保证 -->
    <extension>ppt</extension>
    <mime-type>application/vnd.ms-powerpoint</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>doc</extension>
    <mime-type>application/msword</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>xls</extension>
    <mime-type>application/vnd.ms-excel</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>pps</extension>
    <mime-type>application/vnd.ms-powerpoint</mime-type>
```

```
  </mime-mapping>
  <mime-mapping>
    <extension>MP3</extension>
    <mime-type>audio/x-mpeg</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>mp3</extension>
    <mime-type>audio/x-mpeg</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>wmv</extension>
    <mime-type>video/x-ms-wmv</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>WMV</extension>
    <mime-type>video/x-ms-wmv</mime-type>
  </mime-mapping>
  <welcome-file-list>
<!-- 指定欢迎文件页：指示服务器在收到引用一个目录名而
不是文件名的 URL 时，显示哪个文件作为欢迎页。这里显示 login.jsp -->
    <welcome-file>
            login.jsp
    </welcome-file>
  </welcome-file-list>
  <taglib>    <!--定位 TLD：taglib 元素对标记库描述符文件指定别名-->
    <taglib-uri>dfkj</taglib-uri>
    <taglib-location>/WEB-INF/tlds/dfkj.tld</taglib-location>
  </taglib>
</web-app>
```

9.4.3　系统中的视图设计

在本系统的开发中，首先要完成系统中所有的视图创建。

- 系统主视图及相关视图（如表 9-8 所示）

表 9-8　系统主视图及相关视图表

视图（jsp）	描　　述
alert_info.jsp	信息提示页面
blank.jsp	空白页面
ccframe.jsp	Iframe 页面验证及执行动作
doBeforExit.jsp	退出系统之前执行的操作
doLogin.jsp	执行登录操作
login.jsp	登录页面
main.jsp	主页面
main_index.jsp	任务列表页面

续表

视图（jsp）	描　　述
mainframe.jsp	主框架页面
page_error.jsp	错误信息页面
topframe.jsp	系统菜单页面

❑　与 CASE 有关的视图（如表 9-9 所示）

表 9-9　与 CASE 有关的视图表

视图（jsp）	描　　述
case_close.jsp	Case 关闭页面
case_close_list.jsp	Case 关闭列表页面
case_create.jsp	Case 创建页面
case_deal.jsp	Case 处理页面
case_deal_list.jsp	Case 处理列表页面
case_history_detail.jsp	Case 操作历史页面
case_query_detail.jsp	Case 详情页面
case_query_list.jsp	Case 查询列表页面
case_send.jsp	Case 发送页面
case_send_list.jsp	Case 发送列表页面
case_watch_list.jsp	Case 监控列表页面
hj_add.jsp	呼叫者增加页面
hj_list.jsp	呼叫者管理列表页面
hj_modify.jsp	呼叫者修改页面
knowledge_add.jsp	解决方案增加页面
knowledge_delete.jsp	解决方案修改页面
knowledge_detail.jsp	解决方案详情页面
knowledge_list.jsp	解决方案管理列表页面
knowledge_update.jsp	解决方案修改页面
query_knowledge.jsp	解决方案查询页面
select_case.jsp	选择 Case 页面
select_hj.jsp	选择呼叫者页面
autoalert_add.jsp	自动告警增加页面
autoalert_list.jsp	自动告警配置列表页面
autoalert_update.jsp	自动告警修改页面
manual_alert.jsp	手工告警发送页面
password_updata.jsp	个人密码修改页面
type_add.jsp	系统类别增加页面
type_list.jsp	系统类别管理列表页面
type_modify.jsp	系统类别修改页面
user_add.jsp	用户增加页面

续表

视图（jsp）	描　　述
user_detail.jsp	用户详情页面
user_list.jsp	用户管理列表页面
user_popedom_list.jsp	用户权限列表页面
user_popedom_update.jsp	用户权限修改页面
user_update.jsp	用户修改页面

9.4.4　系统中的包设计

系统中的包设计如图 9-17 所示。

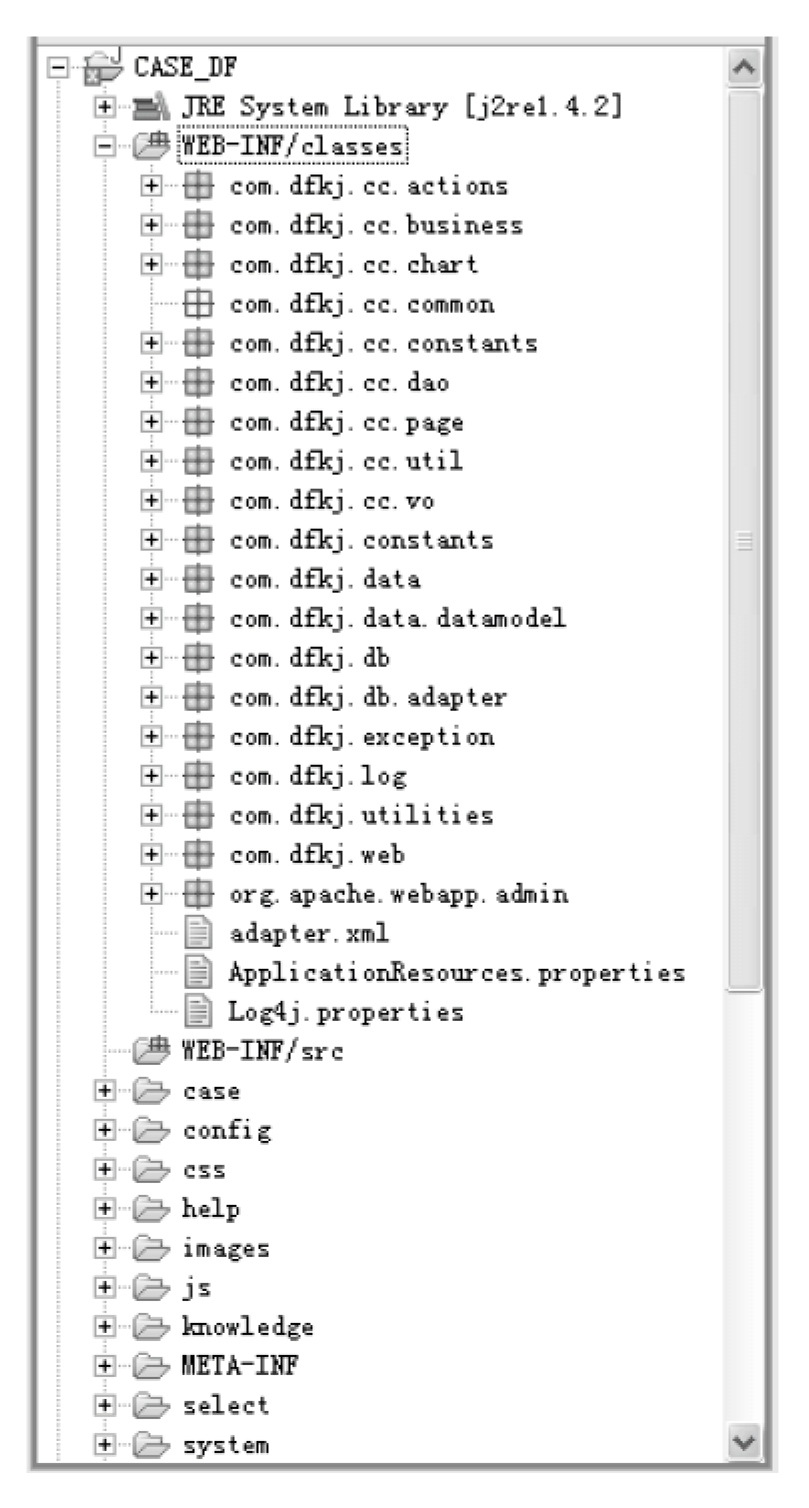


图 9-17　系统包结构图

下面对每个包进行简要说明，如表 9-10 所示。

表 9-10　系统包说明表

包	类	描　　述
CASE\WEB-INF\classes\com\dfkj\cc\actions	所有请求的服务类	
CASE\WEB-INF\classes\com\dfkj\cc\business	业务逻辑/流程的实现类	这里仅与用户有关
CASE\WEB-INF\classes\com\dfkj\cc\constants	本系统所用到的常量类	
CASE\WEB-INF\classes\com\dfkj\cc\dao	访问数据库的操作（增加、删除、查询）类	

续表

包	类	描　述
CASE\WEB-INF\classes\com\dfkj\cc\vo	与数据库基表的映射，即数值对象类	
CASE\WEB-INF\classes\com\dfkj\web	Iaction	实现 Iaction 接口，所有请求服务类的接口
	LoginInfor	登录信息
	MainController	主控制器
	MainFilter	初始化字符集过滤器
	PowerFilter	权限控制
	RequestProcessor	对主控制器补充的一些类
	WebParam	统一定义参数名称
CASE\WEB-INF\classes\com\dfkj\utilities	一些特殊处理的类	
CASE\WEB-INF\classes\com\dfkj\log	日志类	
CASE\WEB-INF\classes\com\dfkj\data	序列生成器等类	
CASE\WEB-INF\classes\com\dfkj\exception	程序中定义的异常类	
CASE\WEB-INF\classes\com\dfkj\db	数据库操作封装、datamodel 等类	

9.4.5　数据库的访问连接设计

在本系统中，对数据库的访问连接是通过一个配置文件 adapter.xml 来设置的。在该配置文件中，可以对多种数据库进行连接设置，连接类型主要包括 3 种：使用连接池、使用单一连接和在应用中使用配置连接。

对于具有高数据访问量的应用来说，一个好的策略就是管理一个连接池。通俗地说，就是将每次创建的数据库连接放在一个“池”中，在连接使用完成时并不急于关闭这个连接。当应用程序需要调用一个数据库连接时，数据库相关的接口通过返回一个重用数据库连接来代替重新创建一个数据库连接，只在没有可用的数据库连接时，才重新创建一个。通过这种方式，应用程序可以减少对数据库连接的操作，尤其在多层环境中，多个客户端可以通过共享少量的物理数据库连接来满足系统需求。

通过系统实际运行，发现使用连接池比不使用连接池的速度要快上 3~5 倍。

数据库连接的适配器等类位于\CASE\WEB-INF\classes\com\dfkj\db 包中。下面是 adapter.xml 文件的代码设计，该文件位于\CASE\WEB-INF\classes 下。

❑　adapter.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by wzyou -->
<adapter>
   <!--"connectionType" Switch used for Connection Type :
       "Pool":使用连接池,
```

```
        "Single":使用单一连接,
        "Select":在应用中使用配置连接 -->
    <connectionType>Single</connectionType>
    <!--"序列号类型"开关用于怎样生成主键值:
        "Sequence": 表示从序列号中获取主键值,
        " Self ": 在数据库中自动增量生成主键值,
        "Java": 通过 Java 类得到主键值--,
<sequenceType>Self</sequenceType>

    <driverAdapter>Oracle</driverAdapter><!--直接连接数据库,这里连接 Oracle
-->
    <driversName id="MYSQL">
        <userName>cc_df</userName>
        <passWord>cc_df</passWord>
        <url>jdbc:mysql://192.168.15.12:3307/tsammy</url>
        <driver>org.gjt.mm.mysql.Driver</driver>
        <adapterName>MysqlDbAdapter</adapterName>
    </driversName>
    <driversName id="Sybase">
        <userName>cc_df</userName>
        <passWord>cc_df</passWord>
        <url>jdbc:sybase:Tds:134.96.66.15:6500?charset=cp936</url>
        <driver>com.sybase.jdbc3.jdbc.SybDriver</driver>
        <adapterName>SybaseDbAdapter</adapterName>
    </driversName>
    <driversName id="Oracle">
        <userName>cc_df</userName>
        <passWord>cc_df</passWord>
        <url>jdbc:oracle:thin:@localhost:ORCL</url>
        <driver>oracle.jdbc.driver.OracleDriver</driver>
        <adapterName>OracleDbAdapter</adapterName>
    </driversName>
    <driversName id="ODBC">
        <userName>cc_df</userName>
        <passWord>cc_df</passWord>
        <url>jdbc:oracle:thin:@localhost:1521:ORA8</url>
        <driver>sun.jdbc.odbc.JdbcODBCDriver</driver>
        <adapterName>OdbcDbAdapter</adapterName>
</driversName>

    <poolAdapter>SybasePool</poolAdapter>  <!--通过数据库连接池连接,
                              这里设置成 Sybase 数据库连接池-->
    <driversName id="MYSQLPOOL">
        <userName>cc_df</userName>
        <passWord>cc_df</passWord>
        <url>jdbc:mysql://127.0.0.1:3307/tsammy</url>
        <driver>org.gjt.mm.mysql.Driver</driver>
        <adapterName>java:comp/env/mysqldb</adapterName>
    </driversName>
```

```
    <driversName id="SybasePool">
        <userName>cc_df</userName>
        <passWord>cc_df</passWord>
        <url>jdbc:sybase:Tds:134.96.5.173:6500?charset=cp936</url>
        <driver>com.sybase.jdbc3.jdbc.SybDriver</driver>
        <adapterName>java:comp/env/SybasePool</adapterName>
    </driversName>
</adapter>
```

9.4.6　业务层设计

从整体上来说，可以把业务逻辑部分再分为两层：数据层逻辑和服务层逻辑。在实现业务层逻辑时，需要尽量保持这两个层次之间的松散耦合。这里分别就数据层和服务层的设计进行分析。

❑　数据层设计

在本系统中没有采用持久化管理，而是使用了 Java 的 JDBC 连接方式来实现。

因为 Java 的 JDBC 连接方式提供的方法很多，步骤也很多。为了实现数据库访问层和逻辑业务层的尽量分离，通过 DAO 和 VO 下的类来封装一些主要的数据库操作，例如事务操作、数据增加、数据删除、数据更新、数据各种查询等操作。

DAO 模式的实现至少需要 3 个部分：

- DAO 工厂类。
- DAO 接口。
- DAO 接口的实现类。

DAO 模式抽象出数据访问方式，业务逻辑组件不需要理会底层的数据库访问，而只专注于业务逻辑的实现。DAO 将数据访问集中在独立的一层，所有的数据访问都由 DAO 对象完成，DAO 分离了数据访问的实现与其他业务逻辑，使得系统更具有可维护性。

DAO 有助于提升系统的可移植性。独立的 DAO 层使得系统能在不同的数据库之间轻易切换，底层的数据库实现对于业务逻辑组件是透明的。数据库移植时仅影响 DAO，不同数据库的切换不会影响业务逻辑组件，因而提高了系统的可复用性。

❑　DAO 工厂类

DAOFactory.class 是一个 DAO 工厂类。代码设计如下：

```
package com.dfkj.cc.dao;
public class DAOFactory
{
    private DAOFactory()
    {
    }
    public static DAOFactory newInstance()
    {
        return new DAOFactory();
    }
    public ICcCaseDAO buildCcCaseDAO()        //CASE 信息
```

```
        {
            return new CcCaseDAOImpl();
        }
        public ICcCaseHdDAO buildCcCaseHdDAO()              //CASE 操作历史
        {
            return new CcCaseHdDAOImpl();
        }
        public ICcFfDAO buildCcFfDAO()                      //CASE 分发
        {
            return new CcFfDAOImpl();
        }
        public ICcGxdlDAO buildCcGxdlDAO()                  //告警队列
        {
            return new CcGxdlDAOImpl();
        }
        public ICcGxsbpzDAO buildCcGxsbpzDAO()              //自动告警配置
        {
            return new CcGxsbpzDAOImpl();
        }
        public ICcHjDAO buildCcHjDAO()                      //呼叫者管理
        {
            return new CcHjDAOImpl();
        }
        public ICcQxDAO buildCcQxDAO()                      //权限
        {
            return new CcQxDAOImpl();
        }
        public ICcUserDAO buildCcUserDAO()                  //用户管理
        {
            return new CcUserDAOImpl();
        }
        public ICcKnowledgeDAO buildCcKnowledgeDAO()        //解决方案管理
        {
            return new CcKnowledgeDAOImpl();
        }
        public ICcTypeDAO buildCcTypeDAO()                  //系统类别管理
        {
            return new CcTypeDAOImpl();
        }
}
```

❑　DAO 接口定义

本系统中 DAO 接口定义类设计如表 9-11 所示。

表 9-11　DAO 接口定义类表

DAO 接口定义类	描　　述
ICcCaseDAO	CASE 信息接口类
ICcCaseHdDAO	CASE 操作历史接口类

续表

DAO 接口定义类	描　述
ICcFfDAO	CASE 分发接口类
ICcGxdlDAO	告警队列接口类
ICcGxsbpzDAO	自动告警配置接口类
ICcHjDAO	呼叫者管理接口类
ICcKnowledgeDAO	解决方案管理接口类
ICcQxDAO	权限接口类
ICcTypeDAO	系统类别管理接口类
ICcUserDAO	用户管理接口类

下面以 CASE 信息表为例进行说明。

ICcCaseDAO（接口定义）类的代码设计如下：

```
package com.dfkj.cc.dao;
import java.sql.Connection;
import com.dfkj.exception.DaoException;
import com.dfkj.cc.vo.CcCaseVO;
import java.util.Collection;

public interface ICcCaseDAO
{
  public void insert(Connection transConn,CcCaseVO ccCaseVO)
    throws DaoException;                //增加记录
  public void update(Connection transConn,CcCaseVO ccCaseVO)
    throws DaoException;                //更新记录
  public void delete(Connection transConn,java.lang.String  caseId)
    throws DaoException;                //删除记录
  public CcCaseVO findByPrimaryKey(Connection transConn,java.lang.String
caseId) throws DaoException;            //通过主键查询记录
  public java.util.Collection findAll(Connection transConn) throws
  DaoException;                         //查询所有记录
  public java.util.Collection findByCondition(Connection
    transConn,java.util.Properties condition) throws DaoException;
                                        //通过条件查询
  public void acceptCase(Connection transConn,java.lang.String caseId)
    throws DaoException;                //接收 CASE
  public void closeCase(Connection transConn,java.lang.String caseId)
    throws DaoException;                //关闭 CASE
  public java.util.Collection findUnclosed(Connection transConn) throws
    DaoException;                       //查找没有关闭的 CASE
}
```

❑　DAO 组件的实现

本系统中 DAO 组件的实现类如表 9-12 所示。

表 9-12　DAO 组件实现类表

DAO 组件实现类	描　　述
CcCaseDAOImpl	CASE 信息实现类
CcCaseHdDAOImpl	CASE 操作历史实现类
CcFfDAOImpl	CASE 分发实现类
CcGxdlDAOImpl	告警队列实现类
CcGxsbpzDAOImpl	自动告警配置实现类
CcHjDAOImpl	呼叫者管理实现类
CcKnowledgeDAOImpl	解决方案管理实现类
CcQxDAOImpl	权限实现类
CcTypeDAOImpl	系统类别管理实现类
CcUserDAOImpl	用户管理实现类

下面以 CASE 信息表为例进行说明。

CcCaseDAOImpl（实现）类的代码设计如下：

```
package com.dfkj.cc.dao;
import java.sql.*;
import com.dfkj.exception.*;
import com.dfkj.data.*;
import com.dfkj.cc.vo.CcCaseVO;
import java.util.*;

public class CcCaseDAOImpl implements  ICcCaseDAO
{
     public void insert(Connection transConn,CcCaseVO ccCaseVO)
       throws DaoException  //增加记录
     {
       Debug.println( this.getClass().getName() + " Insert begina");
       Connection conn = null;
       PreparedStatement statement = null;
       String sql = "";
       try
       {
          conn = transConn;
          //根据所选条件判断数据是否存在
          try
          {
             daoFindSame(conn);
             throw new DuplicateDataException("Primary key already exists");
          }
          catch(ObjectNotFoundException objectNotFoundE)
          {
          }
          sql = " INSERT INTO
              cc_case(case_id,hj_id,gz_jb,gz_nr,case_zt,ff_yj,case_cjr,
```

```
        cj_date,close_date,type,cc_type) VALUES(?,?,?,?,?,?,?,sysdate,
        TO_DATE(?,'YYYY-MM-DD HH24:MI:SS') , ? , ?) ";
        statement = conn.prepareStatement(sql);
        statement.setString(1,DaoUtil.NullToStr(ccCaseVO.getCaseId()));
        statement.setString(2,DaoUtil.NullToStr(ccCaseVO.getHjId()));
        statement.setString(3,DaoUtil.NullToStr(ccCaseVO.getGzJb()));
        statement.setString(4,DaoUtil.NullToStr(ccCaseVO.getGzNr()));
        statement.setString(5,DaoUtil.NullToStr(ccCaseVO.getCaseZt()));
        statement.setString(6,DaoUtil.NullToStr(ccCaseVO.getFfYj()));
        statement.setString(7,DaoUtil.NullToStr(ccCaseVO.getCaseCjr()));
        statement.setString(8,DaoUtil.NullToStr(ccCaseVO.getCloseDate()));
        statement.setString(9,DaoUtil.NullToStr(ccCaseVO.getType()));
        statement.setString(10,DaoUtil.NullToStr(ccCaseVO.getCcType()));

        if (statement.executeUpdate() != 1)
        {
           throw new  DaoException("Error adding row.");
        }
     }
     catch(SQLException e)
     {
        Debug.println(e.getMessage());
        throw new  DaoException("Error executing SQL " + sql + e.getMessage());
     }
     finally
     {
        DBUtil.closeStatement(statement);
     }
     Debug.println( this.getClass().getName() + " Insert end");
   }

   public void update(Connection transConn,CcCaseVO ccCaseVO)
             throws DaoException  //更新记录
   {
     Debug.println( this.getClass().getName() + " Update begin");
     Connection conn = null;
     PreparedStatement statement = null;
     String sql = "";
     try
     {
        conn = transConn;
        sql ="UPDATE cc_case SET case_id=?,hj_id=?,gz_jb=?,gz_nr=?,
            case_zt=?, ff_yj=?,case_cjr=?,close_date=
            TO_DATE(?,'YYYY-MM-DD HH24:MI:SS'),type
             = ? , cc_type = ? WHERE  case_id = ? ";
        statement = conn.prepareStatement(sql);
        statement.setString(1, DaoUtil.NullToStr(ccCaseVO.getCaseId()));
        statement.setString(2, DaoUtil.NullToStr(ccCaseVO.getHjId()));
        statement.setString(3, DaoUtil.NullToStr(ccCaseVO.getGzJb()));
```

```
        statement.setString(4,DaoUtil.NullToStr(ccCaseVO.getGzNr()));
        statement.setString(5,DaoUtil.NullToStr(ccCaseVO.getCaseZt()));
        statement.setString(6,DaoUtil.NullToStr(ccCaseVO.getFfYj()));
        statement.setString(7,DaoUtil.NullToStr(ccCaseVO.getCaseCjr()));
        statement.setString(8,DaoUtil.NullToStr(ccCaseVO.getCloseDate()));
        statement.setString(9,DaoUtil.NullToStr(ccCaseVO.getType()));
        statement.setString(10,DaoUtil.NullToStr(ccCaseVO.getCcType()));
        //条件
        statement.setString(11,DaoUtil.NullToStr(ccCaseVO.getCaseId()));
        if (statement.executeUpdate() != 1)
        {
          throw new  ObjectNotFoundException("Error updating row");
        }
      }
      catch(SQLException e)
      {
         Debug.println(e.getMessage());
         throw new  DaoException("Error executing SQL " + sql +
         e.getMessage());
      }
      finally
      {
        DBUtil.closeStatement(statement);
      }
      Debug.println( this.getClass().getName() + " Update end");
    }
    public void delete(Connection transConn,java.lang.String  caseId)
            throws DaoException  //删除记录
    {
      Debug.println( this.getClass().getName() + " Delete begin");
      Connection conn = null;
      PreparedStatement statement = null;
      try
      {
        conn = transConn;
        statement = conn.prepareStatement("DELETE  cc_case
          WHERE case_id=?");
        //条件
        statement.setString(1, DaoUtil.NullToStr(caseId));
        if (statement.executeUpdate() != 1)
        {
          throw new  RemoveException("Error deleting row");
        }
      }
      catch(SQLException e)
      {
        Debug.println(e.getMessage());
        throw new  DaoException("Error executing SQL  DELETE  cc_case
           WHERE  case_id = ? " + e.getMessage());
```

```
        }
        finally
        {
           DBUtil.closeStatement(statement);
        }
        Debug.println( this.getClass().getName() + " Delete end");
      }

      public CcCaseVO findByPrimaryKey(Connection transConn,java.lang.String
      caseId) throws DaoException  //通过主键查询
      {
        Debug.println( this.getClass().getName() + " Select begin");
        Connection conn = null;
        PreparedStatement statement = null;
        ResultSet rs = null;
        CcCaseVO ccCaseVO = null;
        String sql = "";
        try
        {
           conn = transConn;
           sql=" SELECT case_id,hj_id,gz_jb,gz_nr,case_zt,ff_yj,case_cjr,
           TO_CHAR(cj_date,'YYYY-MM-DD HH24:MI:SS') AS cj_date ,
           TO_CHAR(close_date,'YYYY-MM-DD HH24:MI:SS') AS close_date , type ,
           cc_type FROM  cc_case WHERE  case_id = ? ";
           statement = conn.prepareStatement(sql);
           //条件
           statement.setString(1, DaoUtil.NullToStr(caseId));
           rs = statement.executeQuery();
           if (rs.next())
           {
             ccCaseVO = new CcCaseVO();
             ccCaseVO.setCaseId(rs.getString("case_id"));
             ccCaseVO.setHjId(rs.getString("hj_id"));
             ccCaseVO.setGzJb(rs.getString("gz_jb"));
             ccCaseVO.setGzNr(rs.getString("gz_nr"));
             ccCaseVO.setCaseZt(rs.getString("case_zt"));
             ccCaseVO.setFfYj(rs.getString("ff_yj"));
             ccCaseVO.setCaseCjr(rs.getString("case_cjr"));
             ccCaseVO.setCjDate(rs.getString("cj_date"));
             ccCaseVO.setCloseDate(rs.getString("close_date"));
             ccCaseVO.setType(rs.getString("type"));
             ccCaseVO.setCcType(rs.getString("cc_type"));
           }
           else
           {
              throw new ObjectNotFoundException("Row does not exist.");
           }
        }
        catch(SQLException e)
```

```
        {
          Debug.println(e.getMessage());
          throw new DaoException("Error executing SQL"+sql+e.getMessage());
        }
        finally
        {
          DBUtil.closeResultSet(rs);
          DBUtil.closeStatement(statement);
        }
        Debug.println( this.getClass().getName() + " Select end");
        return ccCaseVO;
      }
      public java.util.Collection findAll(Connection transConn) throws
      DaoException
      {                //查询所有记录
        Debug.println( this.getClass().getName() + " Select begin");
        Connection conn = null;
        PreparedStatement statement = null;
        ResultSet rs = null;
        Vector result = new Vector();
        CcCaseVO ccCaseVO = null;
        String sql = "";
        try
        {
          conn = transConn;
          sql="SELECT case_id,hj_id,gz_jb,gz_nr, case_zt , ff_yj , case_cjr ,
          TO_CHAR(cj_date,'YYYY-MM-DD HH24:MI:SS') AS cj_date ,
          TO_CHAR(close_date,'YYYY-MM-DD HH24:MI:SS') AS close_date , type ,
          cc_type FROM  cc_case ORDER BY case_id ASC ";
          statement = conn.prepareStatement(sql);
          rs = statement.executeQuery();
          while (rs.next())
          {
            ccCaseVO = new CcCaseVO();
            ccCaseVO.setCaseId(rs.getString("case_id"));
            ccCaseVO.setHjId(rs.getString("hj_id"));
            ccCaseVO.setGzJb(rs.getString("gz_jb"));
            ccCaseVO.setGzNr(rs.getString("gz_nr"));
            ccCaseVO.setCaseZt(rs.getString("case_zt"));
            ccCaseVO.setFfYj(rs.getString("ff_yj"));
            ccCaseVO.setCaseCjr(rs.getString("case_cjr"));
            ccCaseVO.setCjDate(rs.getString("cj_date"));
            ccCaseVO.setCloseDate(rs.getString("close_date"));
            ccCaseVO.setType(rs.getString("type"));
            ccCaseVO.setCcType(rs.getString("cc_type"));
            result.add(ccCaseVO);
          }
        }
        catch(SQLException e)
```

```
        {
          Debug.println(e.getMessage());
          throw new DaoException("Error executing SQL"+sql+e.getMessage());
        }
        finally
        {
          DBUtil.closeResultSet(rs);
          DBUtil.closeStatement(statement);
        }
        Debug.println( this.getClass().getName() + " Select end");
        return result ;
      }
      public java.util.Collection findByCondition(Connection transConn,java.
util.Properties condition) throws DaoException        //通过条件查询
      {
          Debug.println( this.getClass().getName() + " Select begin");
          Connection conn = null;
          PreparedStatement statement = null;
          ResultSet rs = null;
          Vector result = new Vector();
          CcCaseVO ccCaseVO = null;
          String sql = "";
          sql="SELECT a.case_id,a.hj_id,gz_jb,gz_nr,case_zt,ff_yj,case_cjr ,
          TO_CHAR(cj_date,'YYYY-MM-DD HH24:MI:SS') AS cj_date ,
          TO_CHAR(close_date,'YYYY-MM-DD HH24:MI:SS') AS close_date , type ,
          cc_type FROM  cc_case a, cc_hj b, cc_user d ";
          String whereClause ="WHERE a.hj_id = b.hj_id AND a.case_cjr
          =d.user_name ";
         String orderClause = " ORDER BY case_id ASC ";
         String fieldValue = null;
         //输出查询条件
          if (condition != null)
          {
            fieldValue = condition.getProperty("HJ_XM");
            if( fieldValue != null && fieldValue.length() > 0)
            {
              whereClause += " AND b.hj_xm LIKE '%"+fieldValue+"%'";
            }
            fieldValue = condition.getProperty("HJ_GS");
            if( fieldValue != null && fieldValue.length() > 0)
            {
              whereClause += " AND b.hj_gs LIKE '%"+fieldValue+"%'";
            }
            fieldValue = condition.getProperty("HJ_DEPT");
            if( fieldValue != null && fieldValue.length() > 0)
            {
              whereClause += " AND b.hj_dept LIKE '%"+fieldValue+"%'";
            }
            fieldValue = condition.getProperty("HJ_DH");
```

```
if( fieldValue != null &&  fieldValue.length() > 0)
{
  whereClause += " AND b.hj_dh LIKE '%"+fieldValue+"%'";
}
fieldValue = condition.getProperty("CASE_ID");
if( fieldValue != null &&  fieldValue.length() > 0)
{
  whereClause += " AND a.case_id LIKE '%"+fieldValue+"%'";
}
fieldValue = condition.getProperty("CASE_CJR");
if( fieldValue != null &&  fieldValue.length() > 0)
{
  whereClause += " AND d.name LIKE '%"+fieldValue+"%'";
}
fieldValue = condition.getProperty("BEGIN_CJDATE");
if( fieldValue != null &&  fieldValue.length() > 0)
{
 whereClause += " AND a.cj_date >=
 TO_DATE('"+fieldValue.substring(0,10)+"00:00:00"+"','YYYY-MM-DD
 HH24:MI:SS') ";
}
fieldValue = condition.getProperty("END_CJDATE");
if( fieldValue != null &&  fieldValue.length() > 0)
{
 whereClause += " AND a.cj_date <=
 TO_DATE('"+fieldValue.substring(0,10)+" 23:59:59"+"',
 'YYYY-MM-DD HH24:MI:SS') ";
}
fieldValue = condition.getProperty("CASE_ZT");
if( fieldValue != null &&  fieldValue.length() > 0)
{
  whereClause += " AND a.case_zt = '"+fieldValue+"'";
}
fieldValue = condition.getProperty("GZ_JB");
if( fieldValue != null &&  fieldValue.length() > 0)
{
  whereClause += " AND a.gz_jb = '"+fieldValue+"'";
}
fieldValue = condition.getProperty("BEGIN_CLOSEDATE");
if( fieldValue != null &&  fieldValue.length() > 0)
{
 whereClause += " AND a.close_date >=
 TO_DATE('"+fieldValue.substring(0,10)+"
 00:00:00"+"','YYYY-MM-DD HH24:MI:SS') ";
}
fieldValue = condition.getProperty("END_CLOSEDATE");
if( fieldValue != null &&  fieldValue.length() > 0)
{
 whereClause += " AND a.close_date <=
```

```
              TO_DATE('"+fieldValue.substring(0,10)+"
              23:59:59"+"','YYYY-MM-DD HH24:MI:SS') ";
            }
            fieldValue = condition.getProperty("CASE_DEAL");
            if( fieldValue != null && fieldValue.length() > 0)
            {
              sql += ", cc_ff e ";
              whereClause = whereClause + " AND a.case_id  = e.case_id AND
              e.cl_ren = '" + condition.getProperty("USER_NAME") + "'";
            }
            fieldValue = condition.getProperty("CASE_CLOSE");
            if( fieldValue != null && fieldValue.length() > 0)
            {
              whereClause += " AND upper(a.case_zt) != 'CLOSE' ";
            }
            fieldValue = condition.getProperty("TYPE");
            if( fieldValue != null && fieldValue.length() > 0)
            {
              whereClause += " AND a.type = '"+fieldValue+"'";
            }
            fieldValue = condition.getProperty("CC_TYPE");
            if( fieldValue != null && fieldValue.length() > 0)
            {
              whereClause += " AND a.cc_type = '"+fieldValue+"'";
            }
            fieldValue = condition.getProperty("ORDER_TYPE");
            if( fieldValue != null && fieldValue.length() > 0)
            {
              orderClause = " ORDER BY case_id DESC ";
            }
          }
          sql += whereClause;
          sql += orderClause;
          //查询条件完毕
        try
        {
          conn = transConn;
          statement = conn.prepareStatement(sql);
          rs = statement.executeQuery();
          while (rs.next())
          {
            ccCaseVO = new CcCaseVO();
            ccCaseVO.setCaseId(rs.getString("case_id"));
            ccCaseVO.setHjId(rs.getString("hj_id"));
            ccCaseVO.setGzJb(rs.getString("gz_jb"));
            ccCaseVO.setGzNr(rs.getString("gz_nr"));
            ccCaseVO.setCaseZt(rs.getString("case_zt"));
            ccCaseVO.setFfYj(rs.getString("ff_yj"));
            ccCaseVO.setCaseCjr(rs.getString("case_cjr"));
```

```
            ccCaseVO.setCjDate(rs.getString("cj_date"));
            ccCaseVO.setCloseDate(rs.getString("close_date"));
            ccCaseVO.setType(rs.getString("type"));
            ccCaseVO.setCcType(rs.getString("cc_type"));
            result.add(ccCaseVO);
          }
      }
      catch(Exception e)
      {
        Debug.println(e.getMessage());
        throw new DaoException("Error executing SQL" +sql+e.getMessage());
      }
      Debug.println( this.getClass().getName() + " Select end");
      return result ;
    }
    public void acceptCase(Connection transConn,java.lang.String  caseId)
      throws DaoException      //接收 CASE
    {
      Debug.println( this.getClass().getName() + " Update begin");
      Connection conn = null;
      PreparedStatement statement = null;
      String sql = "";
      try
      {
         conn = transConn;
         sql = " UPDATE cc_case SET case_zt = 'Receive' WHERE  case_id=? ";
         statement = conn.prepareStatement(sql);
         //条件
         statement.setString(1, DaoUtil.NullToStr(caseId));
         if (statement.executeUpdate() != 1)
         {
            throw new  ObjectNotFoundException("Error updating row");
         }
      }
      catch(SQLException e)
      {
         Debug.println(e.getMessage());
         throw new  DaoException("Error executing SQL " + sql + e.getMessage());
      }
      finally
      {
         DBUtil.closeStatement(statement);
      }
      Debug.println( this.getClass().getName() + " Update end");
    }

    public void closeCase(Connection transConn,java.lang.String  caseId)
           throws DaoException    //关闭 CASE
    {
```

```
    Debug.println( this.getClass().getName() + " Update begin");

    Connection conn = null;
    PreparedStatement statement = null;
    String sql = "";
    try
    {
       conn = transConn;
       sql = " UPDATE  cc_case SET case_zt = 'Close' , close_date = sysdate
         WHERE  case_id = ? ";
       statement = conn.prepareStatement(sql);
       //条件
       statement.setString(1, DaoUtil.NullToStr(caseId));
       if (statement.executeUpdate() != 1)
       {
          throw new  ObjectNotFoundException("Error updating row");
       }
    }
    catch(SQLException e)
    {
       Debug.println(e.getMessage());
       throw new DaoException("Error executing SQL"+sql+e.getMessage());
    }
    finally
    {
       DBUtil.closeStatement(statement);
    }
    Debug.println( this.getClass().getName() + " Update end");
  }
  private void daoFindSame(Connection transConn) throws DaoException
  {
    throw new ObjectNotFoundException("没有发现相同的数据! ");
  }

  public java.util.Collection findUnclosed(Connection transConn) throws
  DaoException {          //查找没有关闭的 CASE
    Debug.println( this.getClass().getName() + " Select begin");
    Connection conn = null;
    PreparedStatement statement = null;
    ResultSet rs = null;
    Vector result = new Vector();
    CcCaseVO ccCaseVO = null;
     String sql = "";
    try
    {
       conn = transConn;
       sql="SELECT case_id,hj_id,gz_jb,gz_nr, case_zt , ff_yj , case_cjr ,
       TO_CHAR(cj_date,'YYYY-MM-DD HH24:MI:SS') AS cj_date ,
       TO_CHAR(close_date,'YYYY-MM-DD HH24:MI:SS') AS close_date , type ,
```

```
            cc_type FROM  cc_case where upper(case_zt) != 'CLOSE'
            ORDER BY case_id ASC ";
            statement = conn.prepareStatement(sql);
            rs = statement.executeQuery();
            while (rs.next())
            {
              ccCaseVO = new CcCaseVO();
              ccCaseVO.setCaseId(rs.getString("case_id"));
              ccCaseVO.setHjId(rs.getString("hj_id"));
              ccCaseVO.setGzJb(rs.getString("gz_jb"));
              ccCaseVO.setGzNr(rs.getString("gz_nr"));
              ccCaseVO.setCaseZt(rs.getString("case_zt"));
              ccCaseVO.setFfYj(rs.getString("ff_yj"));
              ccCaseVO.setCaseCjr(rs.getString("case_cjr"));
              ccCaseVO.setCjDate(rs.getString("cj_date"));
              ccCaseVO.setCloseDate(rs.getString("close_date"));
              ccCaseVO.setType(rs.getString("type"));
              ccCaseVO.setCcType(rs.getString("cc_type"));
              result.add(ccCaseVO);
            }
          }
          catch(SQLException e)
          {
            Debug.println(e.getMessage());
            throw new DaoException("Error executing SQL"+sql+e.getMessage());
          }
          finally
          {
            DBUtil.closeResultSet(rs);
            DBUtil.closeStatement(statement);
          }
          Debug.println( this.getClass().getName() + " Select end");
          return result ;
        }
}
```

❑　VO（数值对象）映射

本系统中 VO 映射类如表 9-13 所示。

表 9-13　VO 映射类表

VO 映射类	描　　述
CcCaseHdVO	CASE 操作历史 VO 类
CcCaseVO	CASE 信息 VO 类
CcFfVO	CASE 分发 VO 类
CcGxdlVO	CASE 告警队列 VO 类
CcGxsbpzVO	CASE 自动告警配置 VO 类
CcHjVO	CASE 呼叫 VO 类

续表

VO 映射类	描　述
CcKnowledgeVO	解决方案 VO 类
CcLogInfoVO	登录信息 VO 类
CcQxVO	权限 VO 类
CcTypeVO	系统类别 VO 类

下面以 CASE 信息为例进行设计说明。

CcCaseVO 类的代码设计如下：

```
package com.dfkj.cc.vo;
public class CcCaseVO implements java.io.Serializable
{
     public CcCaseVO()
     {
     }
     private String caseId = "";
     private String hjId = "";
     private String gzJb = "";
     private String gzNr = "";
     private String caseZt = "";
     private String ffYj = "";
     private String caseCjr = "";
     private String cjDate = "";
     private String closeDate = "";
     private String type = "";
     private String ccType = "";
     public void setCaseId(String caseId)      //设置 CASE 流水号
     {
        this.caseId = caseId;
     }
     public String getCaseId()                 //获取 CASE 流水号
     {
        return caseId;
     }
     public void setHjId(String hjId)          //设置呼叫者 ID
     {
        this.hjId = hjId;
     }
     public String getHjId()                   //获取呼叫者 ID
     {
        return hjId;
     }
     public void setGzJb(String gzJb)          //设置紧急级别
     {
        this.gzJb = gzJb;
     }
     public String getGzJb()                   //获取紧急级别
```

```
    {
        return gzJb;
    }
    public void setGzNr(String gzNr)          //设置 CASE 内容
    {
        this.gzNr = gzNr;
    }
    public String getGzNr()                   //获取 CASE 内容
    {
        return gzNr;
    }
    public void setCaseZt(String caseZt)      //设置 CASE 状态
    {
        this.caseZt = caseZt;
    }
    public String getCaseZt()                 //获取 CASE 状态
    {
        return caseZt;
    }
    public void setFfYj(String ffYj)          //设置分发意见
    {
        this.ffYj = ffYj;
    }
    public String getFfYj()                   //获取分发意见
    {
        return ffYj;
    }
    public void setCaseCjr(String caseCjr)    //设置 CASE 创建人
    {
        this.caseCjr = caseCjr;
    }
    public String getCaseCjr()                //获取 CASE 创建人
    {
        return caseCjr;
    }
    public void setCjDate(String cjDate)      //设置创建日期
    {
        this.cjDate = cjDate;
    }
    public String getCjDate()                 //获取创建日期
    {
        return cjDate;
    }
    public void setCloseDate(String closeDate)//设置关闭日期
    {
        this.closeDate = closeDate;
    }
    public String getCloseDate()              //获取关闭日期
    {
```

```
        return closeDate;
    }
    public String getCcType()              //设置 CASE 分类
    {
        return ccType;
    }
    public void setCcType(String ccType)   //获取 CASE 分类
{
        this.ccType = ccType;
    }
}
```

❑ 服务层设计

在本系统中采用 Struts 来构造系统的框架，因此所有的逻辑都是通过 Actions 和 Business 组合起来的，这里将 Business 合并到 Actions 中。

本系统中服务层设计类如表 9-14 所示。

表 9-14　服务层类表

服 务 层 类	描　　述
UserUpdateAction	用户修改类
UserQueryAction	用户查询类
UserPopedomUpdateAction	用户权限修改类
UserAddAction	用户增加类
TypeUpdateAction	系统类别修改类
TypeQueryAction	系统类别查询类
TypeAddAction	系统类别增加类
PasswordUpdateAction	个人密码修改类
ManualAlertAction	手工告警类
KnowledgeUpdateAction	解决方案修改类
KnowledgeQueryAction	解决方案查询类
KnowledgeDeleteAction	解决方案删除类
KnowledgeAddAction	解决方案增加类
HjUpdateAction	呼叫者修改类
HjQueryAction	呼叫者查询类
HjAddAction	呼叫者增加类
CaseWatchAction	CASE 监控类
CaseSendAction	CASE 发送类
CaseQueryAction	CASE 查询类
CaseDealAction	CASE 处理类
CaseCreateAction	CASE 创建类
CaseCloseAction	CASE 关闭类
AutoAlertUpdateAction	自动告警配置修改类
AutoAlertQueryAction	自动告警配置查询类

续表

服 务 层 类	描　　述
AutoAlertDeleteAction	自动告警配置删除类
AutoAlertAddAction	自动告警配置增加类
AutoAlert	自动告警类
AuthenticationAction	CASE 系统登录验证类

下面仅以 CASE 信息流程管理部分的 CASE 创建为例进行说明。

CASE 创建（CaseCreateAction），代码如下：

```
package com.dfkj.cc.actions;
import java.sql.Connection;
import java.util.Properties;
import java.util.Vector;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import com.dfkj.cc.dao.DAOFactory;
import com.dfkj.cc.dao.ICcCaseDAO;
import com.dfkj.cc.dao.ICcCaseHdDAO;
import com.dfkj.cc.dao.ICcHjDAO;
import com.dfkj.cc.dao.ICcKnowledgeDAO;
import com.dfkj.cc.vo.CcCaseHdVO;
import com.dfkj.cc.vo.CcCaseVO;
import com.dfkj.cc.vo.CcKnowledgeVO;
import com.dfkj.cc.vo.CcLogInfoVO;
import com.dfkj.constants.Constants;
import com.dfkj.data.DBUtil;
import com.dfkj.exception.SystemException;
import com.dfkj.cc.util.Property;
import com.dfkj.web.IAction;

public class CaseCreateAction implements IAction{
    public CaseCreateAction() {
    }
    public void perform(
        HttpServletRequest request,
        HttpServletResponse response,
        ServletContext context,
        Connection connection)
        throws ServletException, SystemException {
        String flag = request.getParameter("flag");
        Vector vResult = new Vector();
        ICcHjDAO ihjDao = DAOFactory.newInstance().buildCcHjDAO();
        ICcKnowledgeDAO iccknowledgeDao =
          DAOFactory.newInstance().buildCcKnowledgeDAO();
```

```
        Vector vCaseType = new Vector();
        vCaseType = Property.getCaseType();
        request.setAttribute("vCaseType",vCaseType);
        if(flag.compareTo("0")==0)
        {
            Vector vCcType = new Vector();
            Vector vCaseGzjb = new Vector();
            vCcType = Property.getCcType();
            vCaseGzjb = Property.getCaseGzjb();
            request.setAttribute("vCcType",vCcType); //设置 CASE 分类
            request.setAttribute("vCaseGzjb",vCaseGzjb);
        }
        else if(flag.compareTo("1")==0)
        {
            CcCaseVO caseVO = new CcCaseVO();
            HttpSession session=((HttpServletRequest)request).getSession
            (true);
           CcLogInfoVO loginInfor =
              (CcLogInfoVO)session.getAttribute(Constants.LOGINFOBEAN);
            String caseId = DBUtil.getCaseSEQ(connection);
            String hjId = request.getParameter("hjId");
            String cctype = request.getParameter("cctype");
            String type = request.getParameter("type");
            String gzJb = request.getParameter("gzJb");
            String gzNr = request.getParameter("gzNr").trim();
            String caseZt = request.getParameter("caseZt");
            String ffYj = request.getParameter("ffYj").trim();
            String caseCjr = loginInfor.getUserName();
            caseVO.setCaseId(caseId);
            caseVO.setHjId(hjId);
            caseVO.setCcType(cctype);
            caseVO.setType(type);
            caseVO.setGzJb(gzJb);
            caseVO.setGzNr(gzNr);
            caseVO.setCaseZt(caseZt);
            caseVO.setFfYj(ffYj);
            caseVO.setCaseCjr(caseCjr);
            ICcCaseDAO icaseDao=
              DAOFactory.newInstance().buildCcCaseDAO();
            ICcCaseHdDAO ihdDao =
              DAOFactory.newInstance().buildCcCaseHdDAO();
           CcCaseHdVO casehdVO = new CcCaseHdVO();
           casehdVO.setCaseId(caseId);
           casehdVO.setClNr("Case 创建成功。");
           casehdVO.setClRen(caseCjr);
           casehdVO.setZtBh(caseZt);
           try{
             //插入 CASE
             icaseDao.insert(connection,caseVO);
```

```
            //插入日志
            ihdDao.insert(connection,casehdVO);
            request.setAttribute("msg","Case["+caseId+"]创建成功！");
            request.setAttribute("forward_url","/MainController.do?
            ActionName=com.dfkj.cc.actions.CaseCreateAction&NextPage=
            /case/case_create.jsp&flag=0");
            }catch(Exception e){
              e.printStackTrace();
              request.setAttribute("msg","Case 创建失败！");
              request.setAttribute("forward_url","/MainController.do?
              ActionName=com.dfkj.cc.actions.CaseCreateAction&NextPage
              =/case/case_create.jsp&flag=0");
          }
      }
      else if(flag.compareTo("2")==0)
      {
          try{
              vResult = (Vector) ihjDao.findAll(connection);
          }
          catch(Exception e){
              e.printStackTrace();
          }
          request.setAttribute("vResult",vResult);
      }
      else if(flag.compareTo("3")==0)
      {
          String hjGs = request.getParameter("hjGs").trim();
          String hjDept = request.getParameter("hjDept").trim();
          String hjXm = request.getParameter("hjXm").trim();
          String orderBy = request.getParameter("orderBy").trim();
          Properties condition = new Properties();
          condition.setProperty("HJ_GS", hjGs);
          condition.setProperty("HJ_DEPT", hjDept);
          condition.setProperty("HJ_XM", hjXm);
          condition.setProperty("ORDER_BY", orderBy);
          try
          {
            vResult=(Vector)ihjDao.findByCondition(connection,condition);
          }
          catch(Exception e){
              e.printStackTrace();
          }
          request.setAttribute("hjGs",hjGs);
          request.setAttribute("hjDept",hjDept);
          request.setAttribute("hjXm",hjXm);
          request.setAttribute("orderBy",orderBy);
          request.setAttribute("vResult",vResult);
      }
      if(flag.compareTo("4")==0)
```

```
        {
            try{
                vResult = (Vector) iccknowledgeDao.findAll(connection);
            }
            catch(Exception e){
                e.printStackTrace();
            }
            request.setAttribute("vResult",vResult);
        }
        else if(flag.compareTo("5")==0)
        {
            String klType = request.getParameter("klType").trim();
            String klTitle = request.getParameter("klTitle").trim();
            String gzMs = request.getParameter("gzMs").trim();
            String jjFa = request.getParameter("jjFa").trim();
            Properties condition = new Properties();
            condition.setProperty("KL_TYPE", klType);
            condition.setProperty("KL_TITLE", klTitle);
            condition.setProperty("GZ_MS", gzMs);
            condition.setProperty("JJ_FA", jjFa);
            try
            {
                vResult=(Vector)iccknowledgeDao.findByCondition
                (connection,condition);
            }
            catch(Exception e){
                e.printStackTrace();
            }
            request.setAttribute("klType",klType);
            request.setAttribute("klTitle",klTitle);
            request.setAttribute("gzMs",gzMs);
            request.setAttribute("jjFa",jjFa);
            request.setAttribute("vResult",vResult);
        }
        else if(flag.compareTo("6")==0)
        {
            String klId = request.getParameter("klId");
            CcKnowledgeVO ccknowledgeVO = new CcKnowledgeVO();
            try
            {
                ccknowledgeVO =
                iccknowledgeDao.findByPrimaryKey(connection, klId);
            }
            catch(Exception e){
                e.printStackTrace();
            }
            request.setAttribute("vResult",ccknowledgeVO);
        }
    }
}
```

9.5　运行与调试本章的案例

将源代码中提供的本章应用程序（整个目录 CASE）复制到本机安装的\Apache Software Foundation\Tomcat 5.5\webapps 路径下。

由于本章提供的案例是在 Oracle 8.17 数据库下开发的，所以读者可以在 Oracle 数据库下创建用户 cc_df，其密码为 cc_df，然后将 case.dmp 直接导入到数据库即可。

另外，读者可以将程序进行简单修改（主要是 DAO），将数据库迁移到 MySQL 5.1。

在 Tomcat 5.5 中要进行调试，跟踪调试信息，可以直接运行 Tomcat 5.5 中的 tomcat5.exe 程序。

❑　配置环境变量

在“我的电脑”上单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，弹出 “系统特性”对话框，选择“高级”选项卡，然后单击“环境变量”按钮，即可编辑系统的环境变量。

❑　设置 Server.xml 文件

Tomcat 主目录/conf 下的 server.xml 文件是对 Web 服务器的配置。找到以下代码：

```
<Connector
       URIEncoding="GBK"
       port="8080"
       redirectPort="8443"
       minSpareThreads="25"
       connectionTimeout="20000"
       uRIEncoding="GBK"
       maxSpareThreads="75"
       maxThreads="150"
       maxHttpHeaderSize="8192">
</Connector>
```

可以将 8080 端口改为喜欢使用的端口，如常见的 80（只要不冲突），以后即可利用该端口访问系统 http://localhost:80/CASE。

如果本章案例应用程序（\CASE 整个目录）不放在\Apache Software Foundation\Tomcat 5.5\webapps 路径下，例如放在开发路径 d:\myproject 下，那么在 server.xml 文件中找到以下代码：

```
<Host
     appBase="webapps"
      name="localhost">
</Host>
```

在其间添加一个<Context>元素：

```
<Context path="/CASE"  reloadable="true"
  docBase=" d:\myproject\CASE"
```

```
   workDir=" d:\myproject\CASE\work" >
</Context>
```

在浏览器中打开 http://localhost:80/CASE 时就会转向 d:\myproject\CASE 下的应用程序。

❑ 设置 web.xml 文件

刚开始调试时，可以对\Tomcat5.5\webapps\CASE\WEB-INF 下的 web.xml 中的代码进行修改，如下所示：

```
<filter>
   <!-- 权限过滤器定义 -->
        <filter-name>com.dfkj.web.PowerFilter</filter-name>
        <filter-class>com.dfkj.web.PowerFilter</filter-class>
        <init-param>
            <param-name>flag</param-name>
            <param-value>off</param-value>  <!--off 表示权限有效，on 无效 -->
        </init-param>
    </filter>
```

将 off 改为 on，这样权限控制将不起作用，输入用户名和密码，可以进入任何模块，主要是为了调试方便。

9.6　小　　结

本章以“处理 CASE”的设计与开发为主线，从系统需求分析、系统总体架构的设计、数据库设计、系统详细设计等这些方面逐步深入分析，较为明晰地讲解了这个系统是如何分析、设计与编程实现的，可综合之前所学的基础知识。

系统结合 Struts 框架、MVC 设计模式、XML 技术等知识，给读者提供了一个真实工程的练习环境。

第 10 章　文件管理系统案例

本章将在第 9 章的基础上通过一个项目“文件管理系统”的设计与开发，描述在 Web 中间件 Tomcat 环境下，如何设计一个比较实用的文件管理系统。

从系统需求分析，到系统的总体架构设计、数据库设计、系统包设计，再到说明系统的关键技术，最后详细讲解系统是如何实现的。

读者在学习本章时不仅要学习整个系统的逐步分析与实现的过程，也要学会一些开发系统的关键技术，如数据分页技术、文件树形结构的处理等，这些都是在实际工程中经常要用到的。

10.1　系统需求分析

文件管理相当于工作中的信息平台。主要用于个人及单位大量信息的分类存放，例如，个人及单位备用的文档、资料与图片、草稿等。

10.1.1　需求概述

本系统要达到以下目标：

- ❑ 对文件目录进行统一管理。
- ❑ 对文件进行统一管理。
- ❑ 对目录及文件的权限进行统一管理。

根据上面的需求，画出文件管理系统的用例图，如图 10-1 所示。

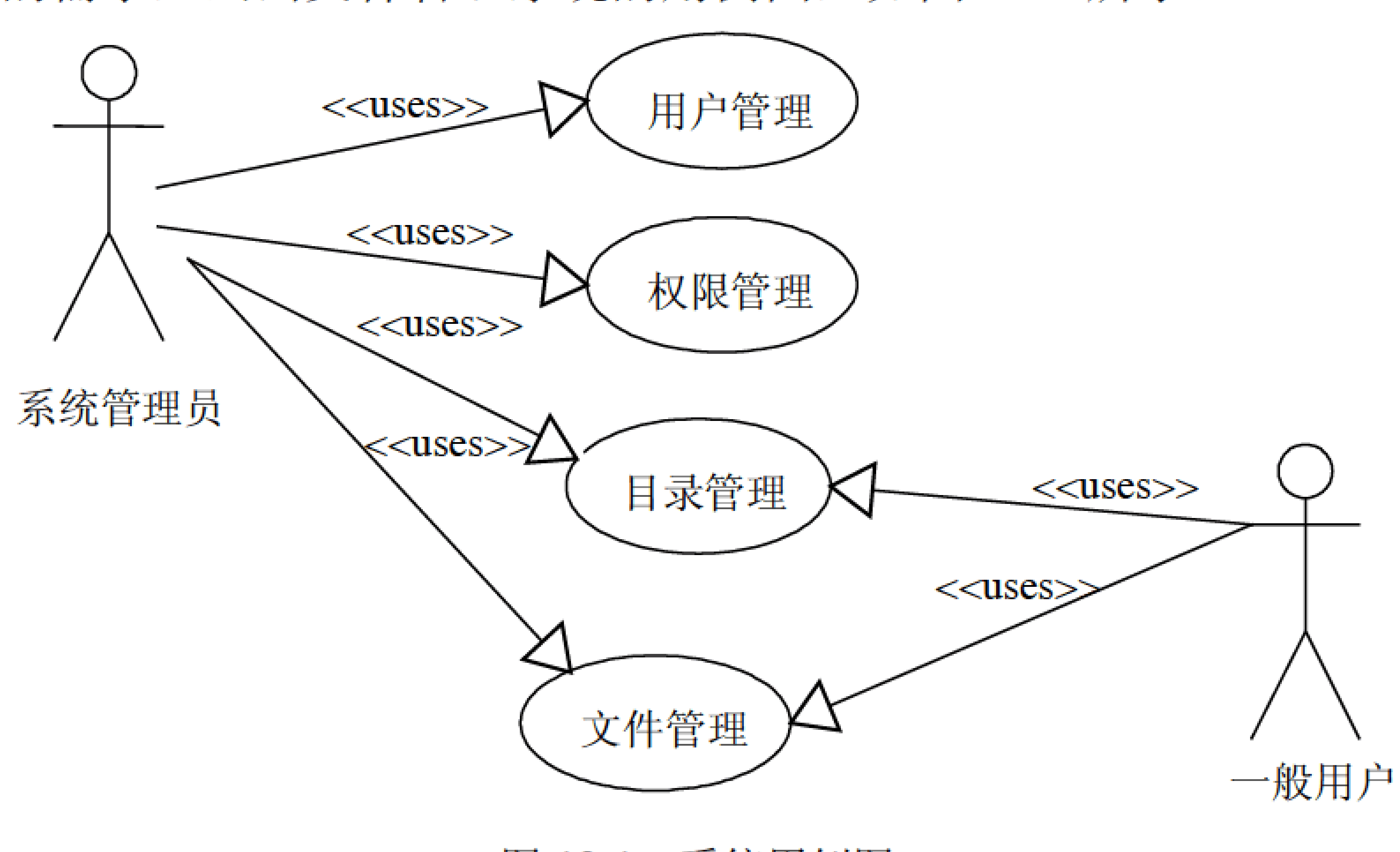


图 10-1　系统用例图

10.1.2　系统功能描述

系统包括目录管理和文件管理等功能。

1. 目录管理

❑　新增目录

增加目录时，系统中将自动增加一个相应的目录。在增加时，要选择上级目录，并输入目录名称及备注，如图 10-2 所示。



图 10-2　新增目录界面

❑　修改目录

提供对目录名称修改的功能，如图 10-3 所示。



图 10-3　修改目录名称界面

❑　删除目录

系统将该目录直接删除。在删除时：

➢　要有提示界面。

➢　目录下不应有文件。

❑　权限设置

给选中的目录进行权限设置，如图 10-4 所示。

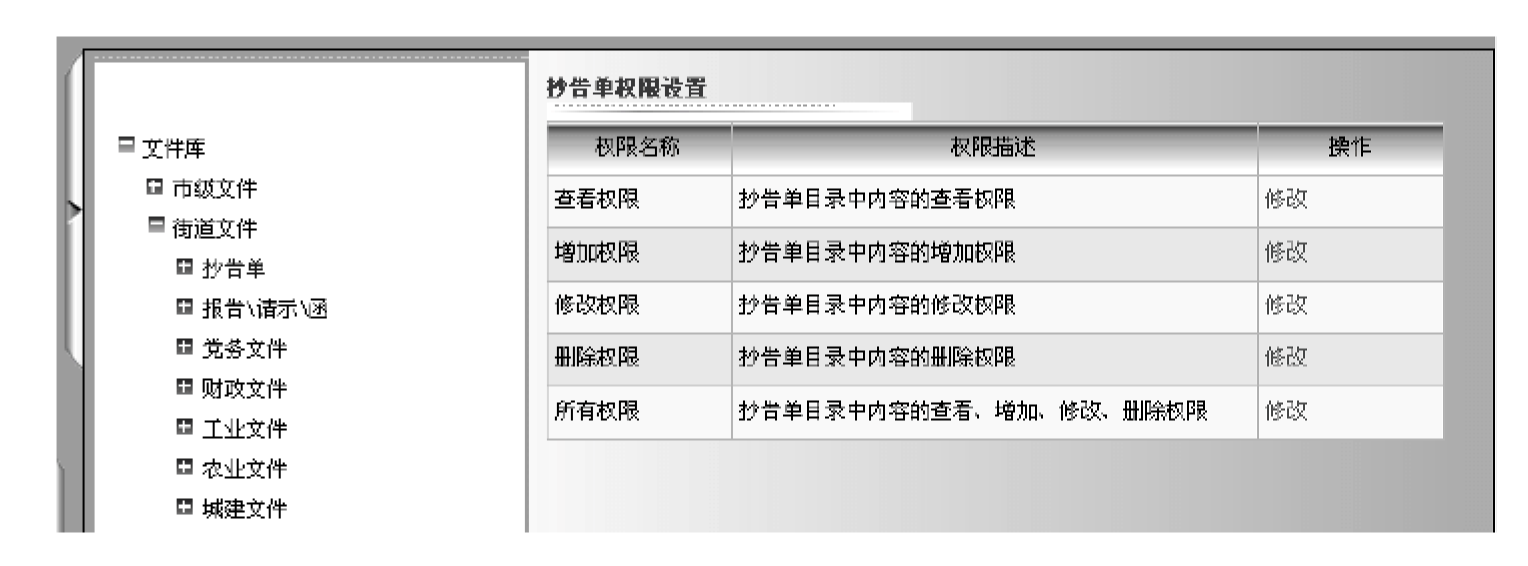


图 10-4　目录权限设置界面

2. 文件管理

❑　新增文件

增加文件，至少输入图 10-5 中文件的基本属性信息，系统中将产生相应的一条新记录。

文件的基本属性包括文档上级目录（只能选择）、文档名称、是否公开此文档（是、否）、本地文档路径（通过“浏览”查找）、备注。



图 10-5　新增文件界面

❑　修改文件

系统允许对文件的名称等属性信息进行修改操作，如图 10-6 所示。



图 10-6　修改文件界面

❑　删除文件

系统根据权限可以删除所添加的文件。删除之前系统要有提示界面。

❑　文件下载

将选中的一个文件下载到本地。可以保存到本地，也可以直接打开。

❑　文件查询

根据输入的文档名称在当前目录下查找该文件。可以模糊查询，如图 10-7 所示。

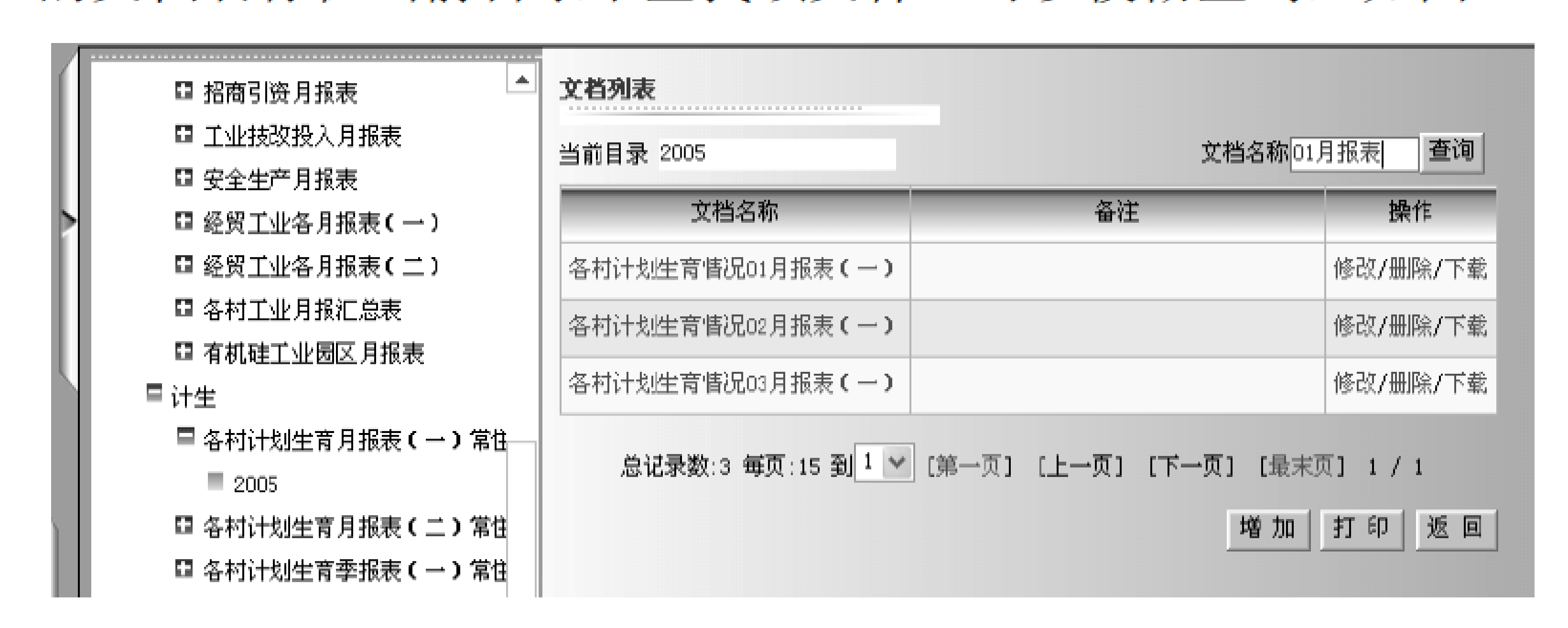


图 10-7　查询文件

10.2　系统总体架构

根据用户的需求，系统功能并不复杂，只有 8 个模块的功能，业务逻辑并不复杂，为快速开发系统，采用了如图 10-8 所示的技术架构。本系统采用 J2EE 的三层结构，分为表现层、业务逻辑层和数据服务层。三层体系将业务规则、数据访问等工作放到中间层处理，客户端不直接与数据库交互，而是通过控制器与中间层建立连接，再由中间层与数据库交互。

系统总体架构如图 10-8 所示。

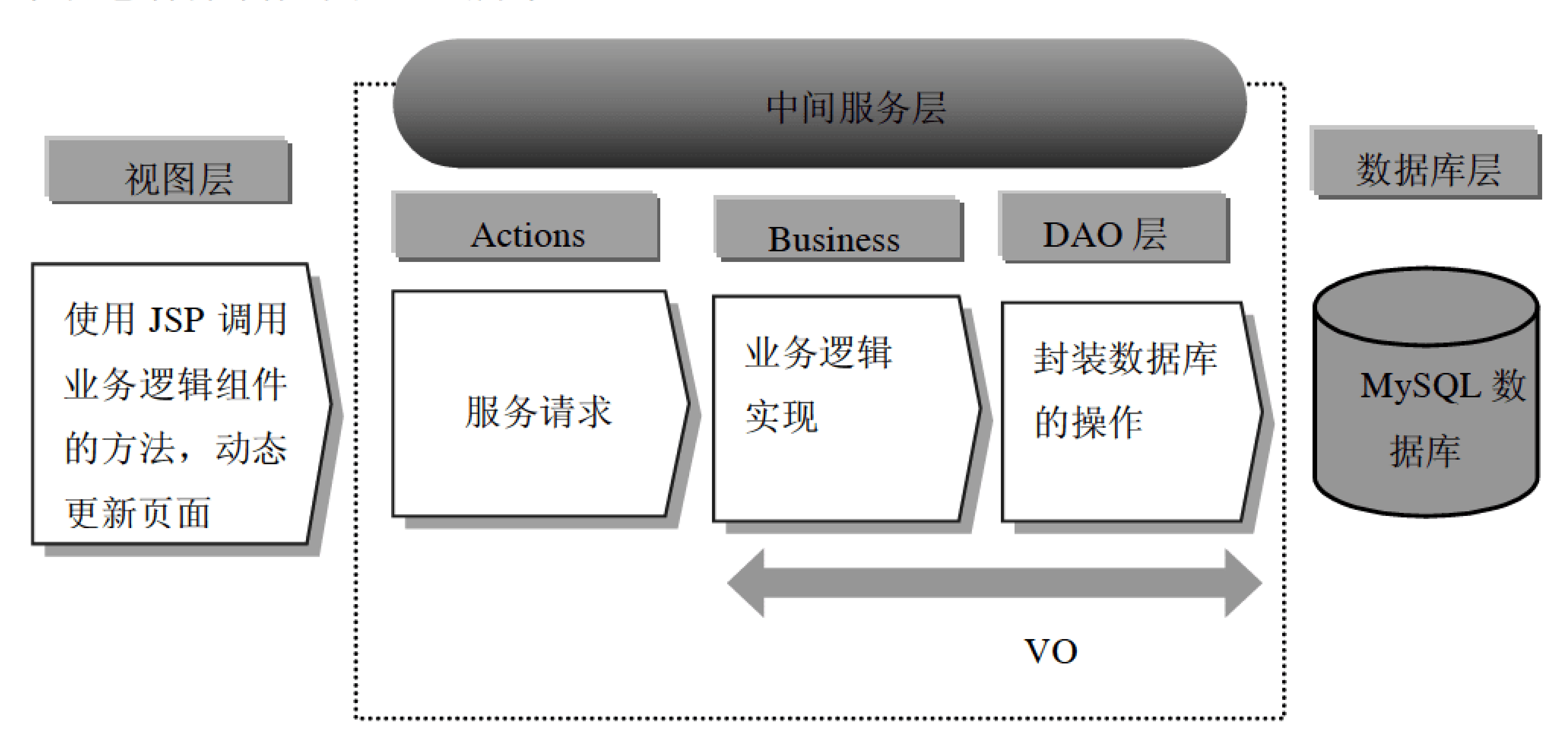


图 10-8　系统的总体设计图

其中：

- Business（业务逻辑）：负责实现业务逻辑，对 DAO 对象进行封装。
- DAO（数据访问对象）：负责与数据库的交互，封装了数据的增加、删除、修改、查询等操作。
- VO（数值对象）：负责与数据库基表的映射。

10.3　数据库设计

数据库的设计一般是先设计出 E-R 图，再根据 E-R 图在数据库中设计出物理的数据表、视图、表与表之间的关系。

10.3.1　E-R 图

一个目录可以有多个文件，不同的目录下可以有相同的文件，因此目录与文件是一对多的关系；一般专门用一张表来存储这种关系。数据库设计的 E-R 图如图 10-9 所示。

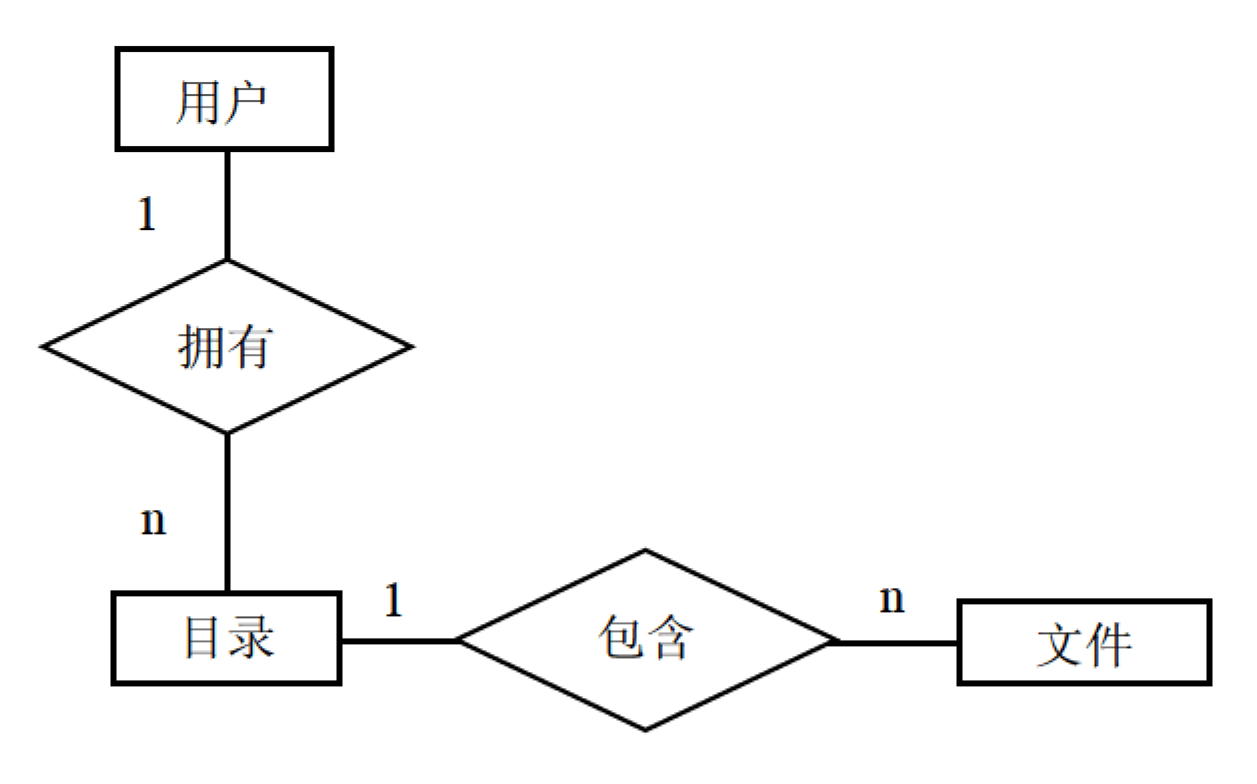


图 10-9　数据库设计 E-R 图

10.3.2　数据模型设计

根据以上的关系来构造系统的数据模型。由于考虑到编程的方便性，本系统将目录、文件及其权限控制放在一个数据库表中。

【特别提示】实际项目开发中，有时根据项目的需要，从编程的角度考虑，需要适当降低范式。

下面建立各个数据模型。具体设计情况如下各表所示。

目录文件信息表（eoa_doc_tree）

表 10-1 主要用来保存目录、文件、目录权限，包括文件/目录节点 ID、文件/目录节点代码、文件/目录节点名称、上一级节点 ID、是否为最后节点、文件/目录名称、是否公开此文件、备注等。

表 10-1　目录文件信息表

字 段 名	中 文 含 义	字 段 类 型	长　度	备　注
DOC_NODE_ID	文件/目录节点 ID	INT	8	
REGION_ID	区域 ID	INT	8	
DOC_NODE_CODE	文件/目录节点代码	VARCHAR	30	
DOC_NODE_NAME	文件/目录节点名称	VARCHAR	80	
PARENT_ID	上一级节点 ID	INT	8	
ISEND	是否为最后节点	VARCHAR	1	
DOC_NAME	文件/目录名称	VARCHAR	200	
ISPUBLIC	是否公开此文件	VARCHAR	2	
REMARK	备注	VARCHAR	80	
FIND_INFO	查询信息	VARCHAR	2000	
ADD_INFO	增加信息	VARCHAR	2000	
MOD_INFO	修改信息	VARCHAR	2000	
DEL_INFO	删除信息	VARCHAR	2000	

与用户、权限等有关的数据库表参见第 11 章。

10.4　系统详细设计

系统详细设计包括界面设计、逻辑主线、系统视图设计、系统包设计、数据库访问连接设计和业务层设计。

10.4.1　界面设计

在本系统的界面设计中，使用 CSS 进行界面设计。CSS 文件在本书所提供的源代码中，在工程文件夹\FILE\css 中有多个 CSS 样式单，这些样式单文件管理页面是最通用的显示效果。当然，读者可以设计自己的 CSS 样式单文件。

\FILE\images 路径中有多个必需的 GIF 文件、JPG 图片文件。当然，读者也可以设计自己喜欢的图片。

具体界面设计参见 10.1.2 节中的各个设计页面。

10.4.2　逻辑主线

一个 Java Web 应用所需要的一个核心文件，就是 web.xml。该文件控制整个应用的行为方式和方法，这是通过各种命令参数的配置来实现的，这些参数在服务启动时自动加载。web.xml 中的元素和 Tomcat 容器完全独立，它是 Web 应用的配置文件。

这里主要介绍贯穿整个系统的 Web 服务基本配置文件 web.xml，其位于\Tomcat 5.5\webapps\FILE\WEB-INF 下。

```
<?xml version="1.0" encoding="UTF-8"?>  <!-- 指定版本和文字编码 -->
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd"> <!-- 指定 DTD 文档的位置
-->
<web-app>
  <display-name>文件管理系统 V1.0</display-name>
            <!-- 标记特定的 Web 应用的名称 -->
  <filter>
    <!--过滤器定义。将一个名字与一个实现 Javax.servlet.Filter 接口类关联。
    这里定义一个过滤器，名为 MainFilter。实现这个过滤器的类是
    com.dfkj.web.MainFilter ->
   <filter-name>com.dfkj.web.MainFilter</filter-name>
   <filter-class>com.dfkj.web.MainFilter</filter-class>
   <init-param>
     <param-name>encoding</param-name>
     <param-value>GBK</param-value>
   </init-param>
   <init-param>
     <param-name>debug_flag</param-name>
```

```
      <param-value>on</param-value>
    </init-param>
  </filter>
  <filter>  <!-- 权限过滤器定义 -->
    <filter-name>com.dfkj.web.PowerFilter</filter-name>
    <filter-class>com.dfkj.web.PowerFilter</filter-class>
    <init-param>
      <param-name>flag</param-name>
      <param-value>off</param-value>   <!--off 表示权限有效，on 无效 -->
    </init-param>
  </filter>
  <filter-mapping>
     <!--主过滤器映射：一旦命名了一个过滤器，就要利用 filter-mapping 元素把它
     与一个或多个 Servlet 或 JSP 页面相关联 -->
    <filter-name>com.dfkj.web.MainFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <filter-mapping>  <!--权限过滤器映射-->
    <filter-name>com.dfkj.web.PowerFilter</filter-name>
    <url-pattern>/MainController.do</url-pattern>
  </filter-mapping>
  <servlet>
<!--Servlet 定义：在向 Servlet 或 JSP 页面制定初始化参数或定制 URL 时，必
须首先命名 Servlet 或 JSP 页面。Servlet 元素就是用来完成此项任务的-->
    <servlet-name>DBServ</servlet-name>
    <display-name>DBServ</display-name>
    <description>用于数据库连接初始化</description>
    <servlet-class>com.dfkj.db.DBServ</servlet-class>
    <load-on-startup>3</load-on-startup>
  </servlet>
  <servlet>
    <!--数据库初始化文件，位于\FILE\WEB-INF\classes-->
    <servlet-name>DbInitConfig</servlet-name>
    <servlet-class>com.dfkj.db.DbInitConfig</servlet-class>
    <init-param>
      <param-name>DbInitFile</param-name>
      <param-value>WEB-INF/classes/adapter.xml</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>Log4jInit</servlet-name>
    <display-name>Log4jInit</display-name>
    <description>Log4j 初始化</description>
    <servlet-class>com.dfkj.log.Log4JInit</servlet-class>
    <init-param>
      <param-name>Log4jInitFile</param-name>
      <param-value>WEB-INF/classes/Log4j.properties</param-value>
    </init-param>
```

```
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
      <!--这里定义了一个名为 MainController 的 Servlet，实现这个 Servlet 的类
      是 com.dfkj.web.MainController-->
    <servlet-name>MainController</servlet-name>
    <display-name>MainController</display-name>
    <description>用于改变字符集</description>
    <servlet-class>com.dfkj.web.MainController</servlet-class>
    <load-on-startup>4</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>validate</param-name>
      <param-value>true</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>treebuilders</param-name>
      <param-value>org.apache.webapp.admin.DepartTreeBuilder</param-value>
    </init-param>
    <init-param>
      <param-name>application</param-name>
      <param-value>ApplicationResources</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>debugjsp</servlet-name>
    <description>编译 JSP 时增加调试信息</description>
    <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
    <init-param>
      <param-name>classdebuginfo</param-name>
      <param-value>true</param-value>
    </init-param>
    <load-on-startup>3</load-on-startup>
```

```
</servlet>
<servlet-mapping>
  <!-- Servlet 映射：服务器一般为 Servlet 提供一个默认的 URL：http://host/
  webAppPrefix/servlet/ServletName。但常常会更改这个 URL，以便 Servlet 可以访问
  初始化参数或更容易地处理相对 URL 时，使用 servlet-mapping 元素-->
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>debugjsp</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>MainController</servlet-name>
  <url-pattern>/MainController.do</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>MainController</servlet-name>
  <url-pattern>/LogoutController.do</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>RegionOutput</servlet-name>
  <url-pattern>/RegionOutput</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>Log4jInit</servlet-name>
  <url-pattern>/servlet/Log4jInit</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>DBServ</servlet-name>
  <url-pattern>/servlet/com.dfkj.db.DBServ</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>DbInitConfig</servlet-name>
  <url-pattern>/servlet/com.dfkj.db.DbInitConfig</url-pattern>
</servlet-mapping>
<session-config>  <!-- 控制会话超时。这里 Session 可以保持不活动状态的最长
                  时间为 60 秒，超过这一时间，Servlet 容器将把它作为无效 Session
                  处理-->
  <session-timeout>60</session-timeout>
</session-config>
<welcome-file-list>
   <!-- 指定欢迎文件页：指示服务器在收到引用一个目录名而不是文件名的 URL 时，显示哪个
   文件作为欢迎页。这里显示 login.jsp -->
  <welcome-file>login.jsp</welcome-file>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
</welcome-file-list>
<taglib>
```

```
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-template.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-template.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/struts</taglib-uri>
    <taglib-location>/WEB-INF/lib/struts.jar</taglib-location>
  </taglib>
  <taglib>   <!--定位 TLD：taglib 元素对标记库描述符文件指定别名-->
    <taglib-uri>dfkj</taglib-uri>
    <taglib-location>/WEB-INF/tlds/dfkj.tld</taglib-location>
  </taglib>
</web-app>
```

10.4.3　系统中的视图设计

在本系统的开发中，首先要完成系统中所有的视图创建。

- ❑ 系统主视图及相关视图（如表 10-2 所示）

表 10-2　系统主视图及相关视图表

视图（jsp）	描　　述
blank.jsp	空白页面
Bottomframe.jsp	底部框架页面
login.jsp	登录页面
logout.jsp	注销页面
main.jsp	登录后的上、中、下框架页面
page_error.jsp	错误提示页面
session_error.jsp	会话错误页面

❑　文件管理部分视图（如表 10-3 所示）

表 10-3　文件管理部分视图表

视图（jsp）	描　　述
ec_documentmanage_main.jsp	文件管理主页面
ec_doc_tree.jsp	树形菜单页面
ec_documentmanage_list.jsp	文件列表页面
ec_documentmanage_add.jsp	增加文件页面
ec_documentmanage_modify.jsp	修改文件页面
transfer_documentmanage.jsp	转换页面
ec_documentmanage_directory_list.jsp	目录列表页面
download.jsp	文件下载页面
ec_documentmanage_directory_add.jsp	增加目录页面
transfer_documentmanage_directory.jsp	删除目录
ec_documentmanage_directory_modify.jsp	目录修改页面
ec_documentpopedom_list.jsp	修改权限页面
ec_documentmanage_viewdoc_main.jsp	浏览文件页面
ec_documentpopedom_modify.jsp	权限修改页面

下面以“树形菜单页面”为例进行设计。弹出式树形菜单如图 10-10 所示。

树形菜单（ec_doc_tree.jsp）代码设计如下：

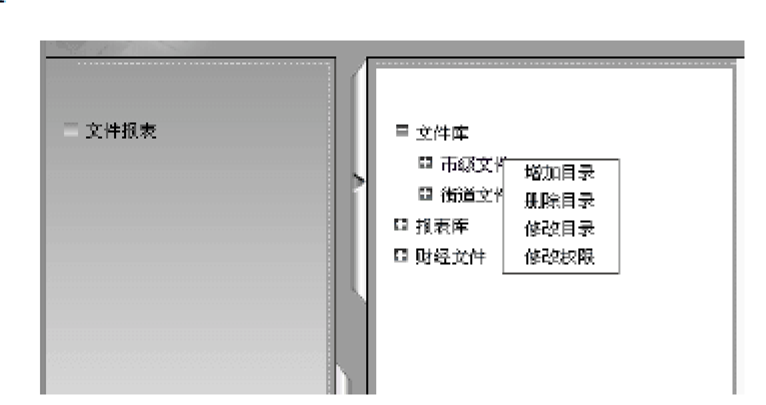


图 10-10　树形菜单页面

```
<%@page import="com.dfkj.eoa.business.eoaDocTreeBuilder"%>
<%@page contentType="text/html;charset=GBK"%>
<html>
<head>
<style>
<!--
#foldheader{cursor:hand ; font-weight:normal ;
list-style-image: url(<%=request.getContextPath()%>/images/fold.gif)}
#foldinglist{cursor:hand ;list-style-image:
   url(<%=request.getContextPath()%>/images/list.gif)}
#doctreeroot{list-style-image:
url(<%=request.getContextPath()%>/images/treeroot.gif)}
#ie5menu
{
   position:absolute;
   width:80px;
   border:1px solid green;
   background-color:menu;
```

```
    font-family:Verdana;
    font-size:9pt;
    line-height:20px;
    cursor:hand;
    visibility:hidden;
}
.menuitems
{
    padding-left:15px;
    padding-right:15px;
}-->
</style>
<script language="JavaScript1.2">
    <!--
    var current_directory_id = -1
    var display_url=0
    function showmenuie5()
    {
    change()
    if (!isEmpty(event.srcElement.name)){
        current_directory_id = event.srcElement.name
    }
    var rightedge=document.body.clientWidth-event.clientX
    var bottomedge=document.body.clientHeight-event.clientY
    if (rightedge<ie5menu.offsetWidth)
    ie5menu.style.left=document.body.scrollLeft+event.
      clientX-ie5menu.offsetWidth
    else
      ie5menu.style.left=document.body.scrollLeft+event.clientX
    if (bottomedge<ie5menu.offsetHeight)
       ie5menu.style.top=document.body.scrollTop+event.
       clientY-ie5menu.offsetHeight
    else
    ie5menu.style.top=document.body.scrollTop+event.clientY
    ie5menu.style.visibility="visible"
    return false
    }
    function hidemenuie5()
    {
    ie5menu.style.visibility="hidden"
    }
    function highlightie5()
    {
    if(event.srcElement.className=="menuitems")
    {
    event.srcElement.style.cursor="hand"
    event.srcElement.style.backgroundColor="highlight"
    event.srcElement.style.color="white"
    if(display_url==1) window.status=event.srcElement.url
    }
```

```
    else
    event.srcElement.style.cursor="default"
    }
    function lowlightie5()
    {
    if (event.srcElement.className=="menuitems")
    {
    event.srcElement.style.backgroundColor=""
    event.srcElement.style.color="black"
    window.status=''
    }
    }
    function jumptoie5()
    {
    if(event.srcElement.name=="add")
    parent.right.location=event.srcElement.url+current_directory_id
    if (current_directory_id == -1)
        return
    if(event.srcElement.name=="del"){
       if(confirm("您确定要删除吗？")){
         parent.right.location=event.srcElement.url+ current_directory_id
       }
    }
    if(event.srcElement.name=="update")
         parent.right.location=event.srcElement.url + current_directory_id
    if(event.srcElement.name=="popedom")
         parent.right.location=event.srcElement.url+current_directory_id
    }
    -->
</script>
<script language="JavaScript1.2">
<!--
var head="display:''"
img1=new Image()
img1.src="<%=request.getContextPath()%>/images/fold.gif"
img2=new Image()
img2.src="<%=request.getContextPath()%>/images/open.gif"
function isEmpty(str)
  {
      if (!str) {
          return true;
      }
      str = str.replace(/^\s+/, "");
      str = str.replace(/\s+$/, "");
      return (str.length == 0);
  }
function directory_modify(){
    parent.right.document.location.href =
    "<%=request.getContextPath()%>/documentmanage
       /ec_documentmanage_directory_list.jsp";
}
```

```
function change(){
    if(!document.all)
        return
    if (event.srcElement.id!="foldheader"&&event.srcElement.id!
                ="foldinglist"){
        return
    }
    i = 0;
    while (document.all[i] != null){
        document.all[i].style.color = ""
        i++
    }//取消所有的显示颜色
    document.all[event.srcElement.sourceIndex].style.color = "blue"
                                        //给当前对象增加颜色
    document.all[event.srcElement.sourceIndex+1].style.color = "black"
                                        //更改当前对象的下属对象为默认颜色
    if (event.srcElement.id=="foldheader") {
        var srcIndex = event.srcElement.sourceIndex
        var nested = document.all[srcIndex+1]
        if (nested.style.display=="none") {
            nested.style.display=''//目录展开
            event.srcElement.style.listStyleImage=
                "url(<%=request.getContextPath()%>/images/open.gif)"
            if (!isEmpty(event.srcElement.name)){
                //alert("11--"+event.srcElement.name);
                parent.right.location.href =
            "<%=request.getContextPath()%>/MainController.do?ActionName=
            com.dfkj.eoa.actions.documentmanage.
            FindEoaDocTreeLeafByParentIdAction&pageSize=
            15&pageNumber=1&NoCache=1&NextPage=/documentmanage/
            ec_documentmanage_list.jsp&parentId="+event.srcElement.name;
            }
        }
        else {
            nested.style.display="none"//目录回缩
            event.srcElement.style.listStyleImage=
                "url(<%=request.getContextPath()%>/images/fold.gif)"
            if (!isEmpty(event.srcElement.name)){
                parent.right.location.href
                ="<%=request.getContextPath()%>/MainController.do?
                ActionName=com.dfkj.eoa.actions.documentmanage.
                 FindEoaDocTreeLeafByParentIdAction&pageSize=
                 15&pageNumber=1&NoCache=1&NextPage=/documentmanage
                 /ec_documentmanage_list.jsp&parentId="
                 +event.srcElement.name;
            }
        }
    }
    if (event.srcElement.id=="foldinglist") {
        if (!isEmpty(event.srcElement.name)){
```

```
            parent.right.location.href =
            "<%=request.getContextPath()%>/MainController.do?
            ActionName=com.dfkj.eoa.actions.documentmanage.
            FindEoaDocTreeLeafByParentIdAction&pageSize=
            15&pageNumber=1&NoCache=1&NextPage=/documentmanage
            /ec_documentmanage_list.jsp&parentId=
            "+event.srcElement.name;
        }
    }
}
function checkpopedom(){
}
document.onclick=change
-->
</script>
<title>树形菜单</title>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<link rel="stylesheet" type="text/css"
      href="<%=request.getContextPath()%>/css/leftmenu.css">
</head>
<body topmargin="0">
<div id="ie5menu" onMouseover="highlightie5()" onMouseout="lowlightie5()"
      onClick="jumptoie5()">
<div class="menuitems" name="add"
url="<%=request.getContextPath()%>/documentmanage/
     ec_documentmanage_directory_add.jsp?docNodeId=">增加目录</div>
<div class="menuitems" name="del"
 url="<%=request.getContextPath()%>/MainController.do?ActionName
=com.dfkj.eoa.actions.documentmanage.DeleteEoaDocTreeAction&NoCache=1
&NextPage=/documentmanage/transfer_documentmanage_directory.jsp&
docNodeId=">删除目录</div>
<div class="menuitems" name="update"
 url="<%=request.getContextPath()%>/MainController.do?ActionName
=com.dfkj.eoa.actions.documentmanage.FindEoaDocTreeByDocNodeIdAction&
NoCache=1&NextPage=/documentmanage/ec_documentmanage_directory_
modify.
jsp&docNodeId=">修改目录</div>
<div class="menuitems" name="popedom"
 url="<%=request.getContextPath()%>/MainController.do?ActionName=
com.dfkj.eoa.actions.documentmanage.FindEoaDocTreeByDocNodeIdAction&
NoCache=1&NextPage=/documentmanage/ec_documentpopedom_list.jsp
&docNodeId=">修改权限</div>
</div>
<script language="JavaScript1.2">
<!--
   document.oncontextmenu=showmenuie5
   if (document.all && window.print) document.body.onclick=hidemenuie5;
-->
</script>
<table width="300px">
```

```
<%
  eoaDocTreeBuilder ecdoctreebuilder = new eoaDocTreeBuilder();
%>
<%
  out.print(ecdoctreebuilder.getTree(request));
%>
</table>
</body>
</html>
```

10.4.4　系统中的包设计

系统中的包设计如图 10-11 所示。

```
FILE
  WEB-INF/classes
    com.dfkj.constants
    com.dfkj.data.datamodel
    com.dfkj.db.adapter
    com.dfkj.eoa.actions.adminwork
    com.dfkj.eoa.actions.deptwork
    com.dfkj.eoa.actions.documentmanage
    com.dfkj.eoa.actions.infogarden
    com.dfkj.eoa.actions.officework
    com.dfkj.eoa.business
    com.dfkj.eoa.common
    com.dfkj.eoa.constants
    com.dfkj.eoa.dao
    com.dfkj.eoa.page
    com.dfkj.eoa.vo
    com.dfkj.exception
    com.dfkj.log
    com.dfkj.pm.actions
    com.dfkj.pm.business
    com.dfkj.pm.constants
    com.dfkj.pm.dao
    com.dfkj.pm.data.datamodel
    com.dfkj.pm.db.adapter
    com.dfkj.pm.exception
    com.dfkj.pm.vo
    com.dfkj.pm.web
    com.dfkj.utilities.page
    com.dfkj.web
    org.apache.webapp.admin
    adapter.xml
    Log4j.properties
```

图 10-11　系统包结构图

下面对每个包进行简要说明，如表 10-4 所示。

表 10-4　包说明表

包	类	描　　述
FILE\WEB-INF\classes\com\dfkj\eoa\actions	所有请求的服务类	
FILE\WEB-INF\classes\com\dfkj\eoa\business	业务逻辑/流程的实现类	
FILE\WEB-INF\classes\com\dfkj\eoa\common	存放通用的类	
FILE\WEB-INF\classes\com\dfkj\eoa\constants	本系统所用到的常量类	
FILE\WEB-INF\classes\com\dfkj\eoa\dao	访问数据库的操作（增加、删除、查询）类	
FILE\WEB-INF\classes\com\dfkj\eoa\vo	与数据库基表的映射，即数值对象类	
FILE\WEB-INF\classes\com\dfkj\web	Iaction.class	实现 Iaction 接口，所有请求服务类的接口

续表

包	类	描　述
FILE\WEB-INF\classes\com\dfkj\web	LoginInfor.class	登录信息
	MainController.class	主控制器
	MainFilter.class	初始化字符集过滤器
	PowerFilter.class	权限控制
	RequestProcessor.class	对主控制器补充的一些类
	WebParam.class	统一定义参数名称
FILE\WEB-INF\classes\com\dfkj\pm	权限部分类	
FILE\WEB-INF\classes\com\dfkj\utilities	分页类	
FILE\WEB-INF\tlds	标签库	
FILE\WEB-INF\classes\com\dfkj\exception	程序中定义的异常类	
FILE\WEB-INF\lib	系统所需要的一些库	

10.4.5　数据库的访问连接设计

在本系统中，对数据库的访问连接是通过一个配置文件 adapter.xml 来设置的。

数据库连接的适配器等类位于\FILE\WEB-INF\classes\com\dfkj\db 包中。当使用连接池时，需要在 Tomcat 中配置好连接池。下面是 adapter.xml 文件的代码设计，该文件位于\FILE\WEB-INF\classes 下。

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter>
<!--连接池开关，使用连接池则 connectionType 项为 Pool,否则可以为其他任意值-->
    <connectionType>file</connectionType>
    <!--单独连接驱动的选择，当 connectionType 不是 Pool 时，
        driverAdapter 在哪个 driversName 前面，则使用该 driver-->
    <driverAdapter>Oracle</driverAdapter>
    <driversName id="MYSQL">
        <userName>mysql</userName>
        <passWord>usemysql</passWord>
        <url>jdbc:mysql://192.168.10.2:3307/file1</url>
        <driver>org.gjt.mm.mysql.Driver</driver>
        <adapterName>MysqlDbAdapter</adapterName>
    </driversName>
    <driversName id="Oracle">
        <userName>file1</userName>
        <passWord>file1</passWord>
        <url>jdbc:oracle:thin:@localhost:1521:orcl</url>
        <driver>oracle.jdbc.driver.OracleDriver</driver>
        <adapterName>OracleDbAdapter</adapterName>
```

```
    </driversName>
    <driversName id="ODBC">
        <userName>wz</userName>
        <passWord>wz</passWord>
        <url>jdbc:oracle:thin:@dell-2900:1521:ORA8</url>
        <driver>sun.jdbc.odbc.JdbcODBCDriver</driver>
        <adapterName>OdbcDbAdapter</adapterName>
    </driversName>
    <!--连接池驱动的选择，当 connectionType 为 Pool 时，poolAdapter 在哪个
      driversName 前面就使用该 driver-->
    <poolAdapter>FILE</poolAdapter>
    <driversName id="MYSQLPOOL">
        <userName>mysql</userName>
        <passWord>usemysql</passWord>
        <url>jdbc:mysql://192.168.10.2:3307/file1</url>
        <driver>org.gjt.mm.mysql.Driver</driver>
        <adapterName>java:comp/env/mysqldb</adapterName>
    </driversName>
</adapter>
```

10.4.6　业务层设计

从整体上来说，可以把业务逻辑部分再分为两层：数据层逻辑和服务层逻辑。本系统在实现业务层逻辑时，需要尽量保持这两个层次之间的松散耦合。这里分别就数据层和服务层的设计进行分析。

- 数据层设计

在本系统中没有采用持久化管理，而是使用了 Java 的 JDBC 连接方式来实现。

因为 Java 的 JDBC 连接方式提供的方法很多，步骤也很多。为了实现数据库访问层和逻辑业务层的尽量分离，通过 DAO 下的类来封装一些主要的数据库操作，例如事务操作、数据增加、数据删除、数据更新、数据各种查询等操作。

DAO 模式的实现至少需要 3 个部分：

- DAO 工厂类。
- DAO 接口。
- DAO 接口的实现类。

3 个部分之间的关系如图 10-12 所示。

DAO 模式抽象出数据访问方式，业务逻辑组件不需要理会底层的数据库访问，而只专注于业务逻辑的实现。DAO 将数据访问集中在独立的一层，所有的数据访问都由 DAO 对象完成，DAO 分离了数据访问的实现与其他业务逻辑，使得系统更具有可维护性。

DAO 有助于提升系统的可移植性。独立的 DAO 层使得系统能在不同的数据库之间轻易切换，底层的数据库实现对于业务逻辑组件是透明的。数据库移植时仅影响 DAO，不同的数据库的切换不会影响业务逻辑组件，因而提高了系统的可复用性。

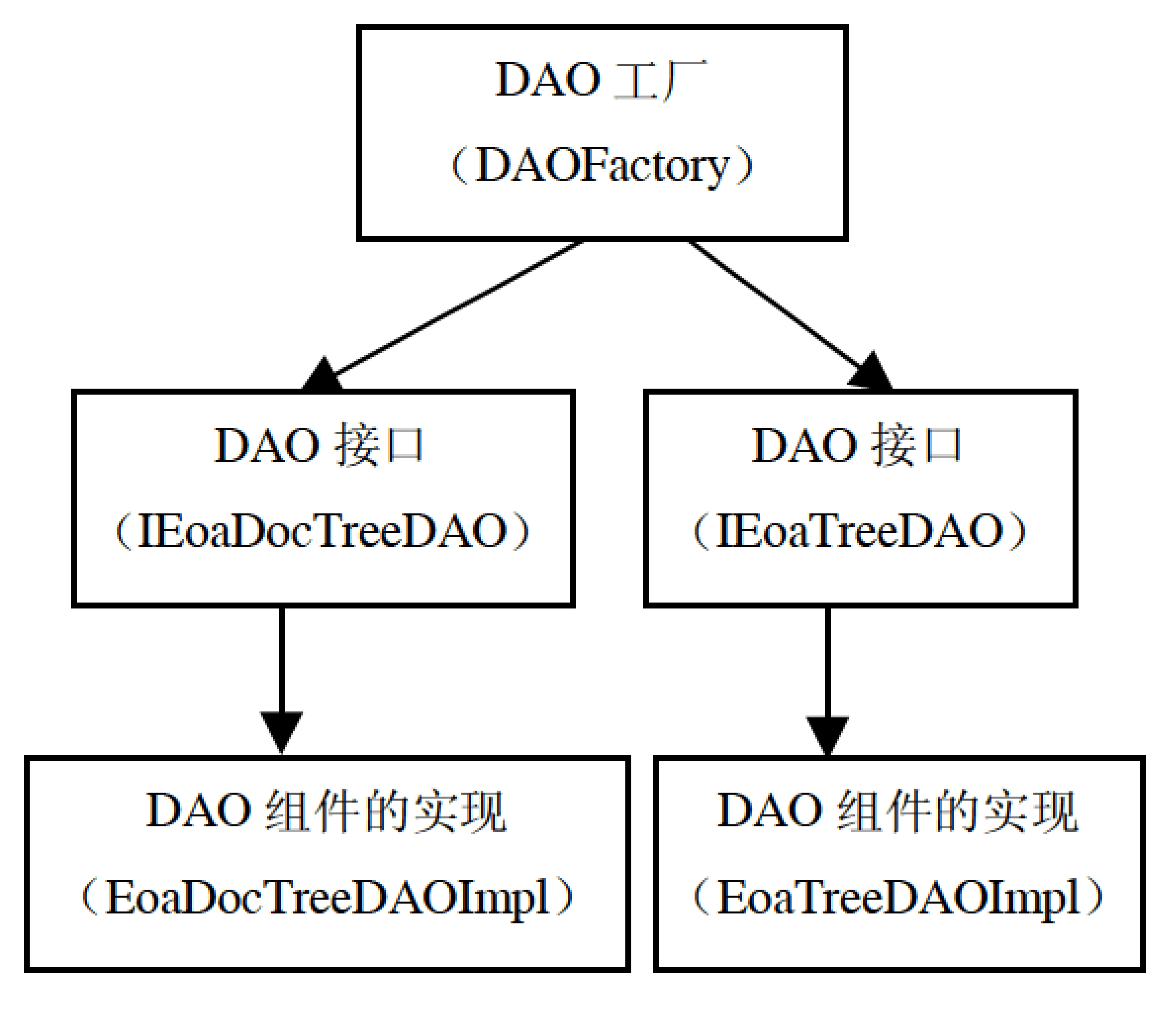


图 10-12　DAO 各部分之间的关系图

❑ DAO 工厂类

DAOFactory.class 是一个 DAO 工厂类。代码设计如下：

```
package com.dfkj.eoa.dao;
import java.sql.Connection;
import com.dfkj.exception.DaoException;
import java.util.Collection;
public class DAOFactory
{
   private DAOFactory()
   {
   }
   public static DAOFactory newInstance()
   {
      return new DAOFactory();
   }
   public IEoaTreeDAO buildEoaTreeDAO()         //树形显示
   {
      return new EoaTreeDAOImpl();
   }
   public IEoaDeptDAO buildEoaDeptDAO()         //部门
   {
      return new EoaDeptDAOImpl();
   }
   public IPUserDAO buildPUserDAO()             //用户
   {
      return new PUserDAOImpl();
   }
   public IEoaDocTreeDAO buildEoaDocTreeDAO()   //文件/目录
   {
       return new EoaDocTreeDAOImpl();
   }
}
```

❑ DAO 接口的定义

本系统中 DAO 接口定义类如表 10-5 所示。

表 10-5　DAO 接口定义类表

DAO 接口定义类	描　　述
IEoaDeptDAO	部门 DAO 定义接口
IPUserDAO	用户 DAO 定义接口
IEoaDocTreeDAO	文件/目录 DAO 定义接口
IEoaTreeDAO	树形 DAO 定义接口

下面以文件/目录信息表进行设计说明。

IEoaDocTreeDAO（接口定义）类的代码设计如下：

```
package com.dfkj.eoa.dao;
import java.sql.Connection;
import com.dfkj.exception.DaoException;
import com.dfkj.eoa.vo.EoaDocTreeVO;
import java.util.Collection;
public interface IEoaDocTreeDAO
{
  public void insert(Connection transConn,EoaDocTreeVO eoaDocTreeVO)
    throws DaoException;                                    //增加文件/目录
  public void update(Connection transConn,EoaDocTreeVO eoaDocTreeVO)
    throws DaoException;                                    //更新文件/目录
  public void delete(Connection transConn,java.lang.String  docNodeId)
    throws DaoException;                                    //删除文件/目录
  public EoaDocTreeVO findByPrimaryKey(Connection transConn,
    java.lang.String  docNodeId) throws DaoException;       //通过主键查询
  public java.util.Collection findAll(Connection transConn) throws
    DaoException;                                           //查询所有文件/目录
  public java.util.Collection findByCondition(Connection transConn,
    java.util.Properties condition) throws DaoException; //通过条件查询文件
  public java.util.Collection findByNodeName(Connection transConn,
    java.lang.String docNodeName ) throws DaoException;  //通过节点名称查询
  public java.util.Collection findByNodeCode(Connection transConn,
    java.lang.String docNodeCode ) throws DaoException;  //通过节点代码查询
  public java.util.Collection findByParentId(Connection transConn,java.
    lang.String parentId ) throws DaoException;             //通过父节点 ID 查询
  public java.util.Collection findByIsEnd(Connection transConn,
    java.lang.String isend ) throws DaoException;           //通过最后节点查询
  public java.util.Collection findByRegionId(Connection transConn,
    java.lang.String regionId ) throws DaoException;        //通过区域 ID 查询
}
```

❑ DAO 组件的实现

本系统中 DAO 组件的实现类如表 10-6 所示。

表 10-6　DAO 组件的实现类表

DAO 组件的实现类	描　　述
EoaDeptDAOImpl	部门 DAO 组件的实现
EoaDocTreeDAOImpl	文件/目录 DAO 组件的实现
EoaTreeDAOImpl	树形 DAO 组件的实现
PUserDAOImpl	用户 DAO 组件的实现

下面以文件/目录信息表为例进行设计说明。

EoaDocTreeDAOImpl（实现）类的代码设计如下：

```
package com.dfkj.eoa.dao;
import java.sql.*;
import com.dfkj.exception.*;
import com.dfkj.data.*;
import com.dfkj.eoa.vo.EoaDocTreeVO;
import java.util.*;
public class EoaDocTreeDAOImpl implements  IEoaDocTreeDAO
{
  public void insert(Connection transConn,EoaDocTreeVO eoaDocTreeVO)
    throws DaoException   //新增文件/目录
  {
    Debug.println( this.getClass().getName() + " Insert begin");
    Connection conn = null;
    PreparedStatement statement = null;
    try
    {
       conn = transConn;
       //根据所选条件判断数据是否存在
       try
       {
          daoFindSame(conn);
          throw new DuplicateDataException("Primary key already exists");
       }
       catch(ObjectNotFoundException objectNotFoundE) {  }
     statement = conn.prepareStatement("INSERT INTO eoa_doc_tree
     (doc_node_id , region_id , doc_node_code , doc_node_name ,
     parent_id , isend , doc_name , ispublic , remark , find_info ,
     add_info , mod_info , del_info )VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?)");
     statement.setString(1,DaoUtil.NullToStr(DBUtil.getSeqFromID
     ("S_DOC_ID",conn)));
     statement.setString(2,DaoUtil.NullToStr(eoaDocTreeVO.getRegionId()));
     statement.setString(3,DaoUtil.NullToStr(eoaDocTreeVO.getDocNodeCode()));
     statement.setString(4,DaoUtil.NullToStr(eoaDocTreeVO.getDocNodeName()));
     statement.setString(5,DaoUtil.NullToStr(eoaDocTreeVO.getParentId()));
     statement.setString(6,DaoUtil.NullToStr(eoaDocTreeVO.getIsend()));
     statement.setString(7,DaoUtil.NullToStr(eoaDocTreeVO.getDocName()));
     statement.setString(8,DaoUtil.NullToStr(eoaDocTreeVO.getIspublic()));
     statement.setString(9,DaoUtil.NullToStr(eoaDocTreeVO.getRemark()));
```

```
        statement.setString(10, DaoUtil.NullToStr(eoaDocTreeVO.getFindInfo()));
        statement.setString(11, DaoUtil.NullToStr(eoaDocTreeVO.getAddInfo()));
        statement.setString(12, DaoUtil.NullToStr(eoaDocTreeVO.getModInfo()));
        statement.setString(13, DaoUtil.NullToStr(eoaDocTreeVO.getDelInfo()));
         if (statement.executeUpdate() != 1)
         {
            throw new  DaoException("Error adding row.");
         }
      }
      catch(SQLException e)
      {
         Debug.println(e.getMessage());
         throw new DaoException("Error executing SQL INSERT INTO
         eoa_doc_tree(doc_node_id , region_id , doc_node_code ,
         doc_node_name , parent_id , isend , doc_name , ispublic , remark,
         find_info , add_info , mod_info , del_info )
         VALUES(?,?,?,?,?,?,?,?,?,?,?,?,? )" + e.getMessage());
      }
      finally
      {
         DBUtil.closeStatement(statement);
      }
      Debug.println( this.getClass().getName() + " Insert end");
   }
   public void update(Connection transConn,EoaDocTreeVO eoaDocTreeVO)
      throws DaoException  //更新文件/目录
   {
      Debug.println( this.getClass().getName() + " Update begin");
      Connection conn = null;
      PreparedStatement statement = null;
      try
      {
         conn = transConn;
         statement = conn.prepareStatement(" UPDATE  eoa_doc_tree SET
         doc_node_id= ? , region_id= ? , doc_node_code= ? , doc_node_name= ? ,
         parent_id = ?, isend= ?, doc_name= ?, ispublic = ? , remark = ? ,
         find_info = ? , add_info = ?  , mod_info = ?  , del_info = ?
         WHERE  doc_node_id = ? ");
       statement.setString(1,DaoUtil.NullToStr(eoaDocTreeVO.getDocNodeId()));
       statement.setString(2, DaoUtil.NullToStr(eoaDocTreeVO.getRegionId()));
       statement.setString(3,DaoUtil.NullToStr(eoaDocTreeVO.getDocNodeCode()));
       statement.setString(4,DaoUtil.NullToStr(eoaDocTreeVO.getDocNodeName()));
       statement.setString(5,DaoUtil.NullToStr(eoaDocTreeVO.getParentId()));
       statement.setString(6,DaoUtil.NullToStr(eoaDocTreeVO.getIsend()));
       statement.setString(7,DaoUtil.NullToStr(eoaDocTreeVO.getDocName()));
       statement.setString(8,DaoUtil.NullToStr(eoaDocTreeVO.getIspublic()));
       statement.setString(9,DaoUtil.NullToStr(eoaDocTreeVO.getRemark()));
       statement.setString(10,DaoUtil.NullToStr(eoaDocTreeVO.getFindInfo()));
       statement.setString(11,DaoUtil.NullToStr(eoaDocTreeVO.getAddInfo()));
```

```
    statement.setString(12,DaoUtil.NullToStr(eoaDocTreeVO.getModInfo()));
    statement.setString(13,DaoUtil.NullToStr(eoaDocTreeVO.getDelInfo()));
      //条件
    statement.setString(14,DaoUtil.NullToStr(eoaDocTreeVO.getDocNodeId()));
      if (statement.executeUpdate() != 1)
      {
         throw new  ObjectNotFoundException("Error updating row");
      }
   }
   catch(SQLException e)
   {
      Debug.println(e.getMessage());
      throw new  DaoException("Error executing SQL  UPDATE
        eoa_doc_tree SET doc_node_id=? ,region_id= ?, doc_node_code = ? ,
        doc_node_name=?,parent_id=?,isend=?,doc_name = ? , ispublic = ? ,
        remark=?,find_info=?,add_info=? ,mod_info=?,del_info= ?
        WHERE  doc_node_id = ?  " + e.getMessage());
   }
   finally
   {
      DBUtil.closeStatement(statement);
   }
   Debug.println( this.getClass().getName() + " Update end");
 }
 public void delete(Connection transConn,java.lang.String  docNodeId)
   throws DaoException   //删除文件/目录
 {
   Debug.println( this.getClass().getName() + " Delete begin");
   Connection conn = null;
   PreparedStatement statement = null;
   try
   {
      conn = transConn;
      statement = conn.prepareStatement(" DELETE  eoa_doc_tree
               WHERE  doc_node_id = ? ");
      //条件
      statement.setString(1, DaoUtil.NullToStr(docNodeId));
      if (statement.executeUpdate() != 1)
      {
         throw new  RemoveException("Error deleting row");
      }
   }
   catch(SQLException e)
   {
      Debug.println(e.getMessage());
      throw new  DaoException("Error executing SQL
      DELETE  eoa_doc_tree WHERE  doc_node_id = ? " + e.getMessage());
   }
   finally
```

```
    {
      DBUtil.closeStatement(statement);
    }
    Debug.println( this.getClass().getName() + " Delete end");
  }
  public EoaDocTreeVO findByPrimaryKey(Connection transConn,java.lang.String
  docNodeId) throws DaoException   //通过主键查询
  {
    Debug.println( this.getClass().getName() + " Select begin");
    Connection conn = null;
    PreparedStatement statement = null;
    ResultSet rs = null;
    EoaDocTreeVO eoaDocTreeVO = null;
    try
    {
      conn = transConn;
      statement = conn.prepareStatement(" SELECT doc_node_id , region_id ,
      doc_node_code , doc_node_name , parent_id , isend ,
      doc_name , ispublic , remark , find_info , add_info , mod_info ,
      del_info FROM  eoa_doc_tree WHERE  doc_node_id = ? ");
      //条件
      statement.setString(1, DaoUtil.NullToStr(docNodeId));
      rs = statement.executeQuery();
      if (rs.next())
      {
        eoaDocTreeVO = new EoaDocTreeVO();
        eoaDocTreeVO.setDocNodeId(rs.getString("doc_node_id"));
        eoaDocTreeVO.setRegionId(rs.getString("region_id"));
        eoaDocTreeVO.setDocNodeCode(rs.getString("doc_node_code"));
        eoaDocTreeVO.setDocNodeName(rs.getString("doc_node_name"));
        eoaDocTreeVO.setParentId(rs.getString("parent_id"));
        eoaDocTreeVO.setIsend(rs.getString("isend"));
        eoaDocTreeVO.setDocName(rs.getString("doc_name"));
        eoaDocTreeVO.setIspublic(rs.getString("ispublic"));
        eoaDocTreeVO.setRemark(rs.getString("remark"));
        eoaDocTreeVO.setFindInfo(rs.getString("find_info"));
        eoaDocTreeVO.setAddInfo(rs.getString("add_info"));
        eoaDocTreeVO.setModInfo(rs.getString("mod_info"));
        eoaDocTreeVO.setDelInfo(rs.getString("del_info"));
      }
      else
      {
        throw new ObjectNotFoundException("Row does not exist.");
      }
    }
    catch(SQLException e)
    {
      Debug.println(e.getMessage());
      throw new  DaoException("Error executing SQL  SELECT doc_node_id ,
```

```
    region_id , doc_node_code , doc_node_name , parent_id , isend ,
    doc_name,ispublic,remark,find_info,add_info,mod_info,del_info
    FROM  eoa_doc_tree WHERE  doc_node_id = ? " + e.getMessage());
  }
  finally
  {
    DBUtil.closeResultSet(rs);
    DBUtil.closeStatement(statement);
  }
  Debug.println( this.getClass().getName() + " Select end");
  return eoaDocTreeVO;
}
public java.util.Collection findAll(Connection transConn) throws DaoException
{  //查询所有的记录
  Debug.println( this.getClass().getName() + " Select begin");
  Connection conn = null;
  PreparedStatement statement = null;
  ResultSet rs = null;
  Vector result = new Vector();
  EoaDocTreeVO eoaDocTreeVO = null;
  try
  {
    conn = transConn;
    statement = conn.prepareStatement(" SELECT doc_node_id , region_id ,
    doc_node_code,doc_node_name,parent_id,isend,doc_name,ispublic ,
    remark,find_info,add_info,mod_info,del_info FROM  eoa_doc_tree");
    rs = statement.executeQuery();
    while (rs.next())
    {
      eoaDocTreeVO = new EoaDocTreeVO();
      eoaDocTreeVO.setDocNodeId(rs.getString("doc_node_id"));
      eoaDocTreeVO.setRegionId(rs.getString("region_id"));
      eoaDocTreeVO.setDocNodeCode(rs.getString("doc_node_code"));
      eoaDocTreeVO.setDocNodeName(rs.getString("doc_node_name"));
      eoaDocTreeVO.setParentId(rs.getString("parent_id"));
      eoaDocTreeVO.setIsend(rs.getString("isend"));
      eoaDocTreeVO.setDocName(rs.getString("doc_name"));
      eoaDocTreeVO.setIspublic(rs.getString("ispublic"));
      eoaDocTreeVO.setRemark(rs.getString("remark"));
      eoaDocTreeVO.setFindInfo(rs.getString("find_info"));
      eoaDocTreeVO.setAddInfo(rs.getString("add_info"));
      eoaDocTreeVO.setModInfo(rs.getString("mod_info"));
      eoaDocTreeVO.setDelInfo(rs.getString("del_info"));
      result.add(eoaDocTreeVO);
    }
  }
  catch(SQLException e)
  {
    Debug.println(e.getMessage());
```

```
            throw new  DaoException("Error executing SQL  SELECT doc_node_id,
              region_id,doc_node_code,doc_node_name,parent_id,isend,doc_name ,
              ispublic , remark , find_info , add_info , mod_info , del_info
              FROM  eoa_doc_tree" + e.getMessage());
        }
        finally
        {
            DBUtil.closeResultSet(rs);
            DBUtil.closeStatement(statement);
        }
        Debug.println( this.getClass().getName() + " Select end");
        return result ;
    }
    public java.util.Collection findByCondition(Connection //通过条件查询
        transConn,java.util.Properties condition) throws DaoException
    {
        Debug.println( this.getClass().getName() + " Select begin");
        Connection conn = null;
        PreparedStatement statement = null;
        ResultSet rs = null;
        Vector result = new Vector();
        Vector fieldList = new Vector();
        String sql = "";
        EoaDocTreeVO eoaDocTreeVO = null;
        try
        {
            conn = transConn;
            sql = " SELECT doc_node_id , region_id , doc_node_code ,
            doc_node_name , parent_id , isend , doc_name , ispublic , remark ,
            find_info , add_info , mod_info , del_info FROM  eoa_doc_tree";
            String whereClause = " WHERE 1=1 ";
            String fieldValue = null;
            //输出查询条件
            if (condition != null)
            {
                fieldValue = condition.getProperty("DOC_NODE_ID");
                if( fieldValue != null && fieldValue.length() > 0)
                {
                    whereClause += " AND doc_node_id = ?  ";
                    fieldList.add(fieldValue);
                }
                fieldValue = condition.getProperty("REGION_ID");
                if( fieldValue != null && fieldValue.length() > 0)
                {
                    whereClause += " AND region_id = ?  ";
                    fieldList.add(fieldValue);
                }
                fieldValue = condition.getProperty("DOC_NODE_CODE");
                if( fieldValue != null && fieldValue.length() > 0)
```

```
        {
          whereClause += " AND doc_node_code =  ?  ";
          fieldList.add(fieldValue);
        }
        fieldValue = condition.getProperty("DOC_NODE_NAME");
        if( fieldValue != null &&  fieldValue.length() > 0)
        {
          whereClause += " AND doc_node_name =  ?  ";
          fieldList.add(fieldValue);
        }
        fieldValue = condition.getProperty("PARENT_ID");
        if( fieldValue != null &&  fieldValue.length() > 0)
        {
          whereClause += " AND parent_id =  ?  ";
          fieldList.add(fieldValue);
        }
        fieldValue = condition.getProperty("ISEND");
        if( fieldValue != null &&  fieldValue.length() > 0)
        {
          whereClause += " AND isend =  ?  ";
          fieldList.add(fieldValue);
        }
        fieldValue = condition.getProperty("DOC_NAME");
        if( fieldValue != null &&  fieldValue.length() > 0)
        {
          whereClause += " AND doc_name =  ?  ";
          fieldList.add(fieldValue);
        }
        fieldValue = condition.getProperty("ISPUBLIC");
        if( fieldValue != null &&  fieldValue.length() > 0)
        {
          whereClause += " AND ispublic =  ?  ";
          fieldList.add(fieldValue);
        }
        fieldValue = condition.getProperty("REMARK");
        if( fieldValue != null &&  fieldValue.length() > 0)
        {
          whereClause += " AND remark =  ?  ";
          fieldList.add(fieldValue);
        }
        fieldValue = condition.getProperty("FIND_INFO");
        if( fieldValue != null &&  fieldValue.length() > 0)
        {
           whereClause += " AND find_info =  ?  ";
           fieldList.add(fieldValue);
         }
        fieldValue = condition.getProperty("ADD_INFO");
        if( fieldValue != null &&  fieldValue.length() > 0)
        {
```

```
            whereClause += " AND add_info =  ?  ";
            fieldList.add(fieldValue);
         }
         fieldValue = condition.getProperty("MOD_INFO");
         if( fieldValue != null &&  fieldValue.length() > 0)
         {
            whereClause += " AND mod_info =  ?  ";
            fieldList.add(fieldValue);
         }
         fieldValue = condition.getProperty("DEL_INFO");
         if( fieldValue != null &&  fieldValue.length() > 0)
         {
            whereClause += " AND del_info =  ?  ";
            fieldList.add(fieldValue);
         }
      }
      sql += whereClause;
      //查询条件完毕
      statement = conn.prepareStatement(sql);
      for(int i=0; i<fieldList.size(); i++)
      {
         statement.setString(i+1,(String)fieldList.elementAt(i));
      }
      rs = statement.executeQuery();
      while (rs.next())
      {
        eoaDocTreeVO = new EoaDocTreeVO();
        eoaDocTreeVO.setDocNodeId(rs.getString("doc_node_id"));
        eoaDocTreeVO.setRegionId(rs.getString("region_id"));
        eoaDocTreeVO.setDocNodeCode(rs.getString("doc_node_code"));
        eoaDocTreeVO.setDocNodeName(rs.getString("doc_node_name"));
        eoaDocTreeVO.setParentId(rs.getString("parent_id"));
        eoaDocTreeVO.setIsend(rs.getString("isend"));
        eoaDocTreeVO.setDocName(rs.getString("doc_name"));
        eoaDocTreeVO.setIspublic(rs.getString("ispublic"));
        eoaDocTreeVO.setRemark(rs.getString("remark"));
        eoaDocTreeVO.setFindInfo(rs.getString("find_info"));
        eoaDocTreeVO.setAddInfo(rs.getString("add_info"));
        eoaDocTreeVO.setModInfo(rs.getString("mod_info"));
        eoaDocTreeVO.setDelInfo(rs.getString("del_info"));
        result.add(eoaDocTreeVO);
      }
   }
   catch(SQLException e)
   {
      Debug.println(e.getMessage());
      throw new  DaoException("Error executing SQL"  + sql +
             e.getMessage());
   }
```

```
    finally
    {
      DBUtil.closeResultSet(rs);
      DBUtil.closeStatement(statement);
    }
    Debug.println( this.getClass().getName() + " Select end");
    return result ;
  }
  private void daoFindSame(Connection transConn) throws DaoException
  {   //查询相同的数据
    throw new ObjectNotFoundException("没有发现相同的数据! ");
  }

public java.util.Collection findByNodeName(Connection transConn,java.lang.
    String docNodeName ) throws DaoException //通过节点名称进行查询
  {
    Debug.println( this.getClass().getName() + " Select begin");
    Connection conn = null;
    PreparedStatement statement = null;
    ResultSet rs = null;
    Vector result = new Vector();
    EoaDocTreeVO eoaDocTreeVO = null;
    try
    {
      conn = transConn;
      statement = conn.prepareStatement(" SELECT doc_node_id , region_id ,
        doc_node_code , doc_node_name , parent_id , isend , doc_name ,
        ispublic , remark , find_info , add_info , mod_info , del_info
     FROM eoa_doc_tree WHERE 1=1 AND doc_node_name=? ORDER BY 1 ");
      //条件
      statement.setString(1, DaoUtil.NullToStr(docNodeName));
      rs = statement.executeQuery();
      while (rs.next())
      {
        eoaDocTreeVO = new EoaDocTreeVO();
        eoaDocTreeVO.setDocNodeId(rs.getString("doc_node_id"));
        eoaDocTreeVO.setRegionId(rs.getString("region_id"));
        eoaDocTreeVO.setDocNodeCode(rs.getString("doc_node_code"));
        eoaDocTreeVO.setDocNodeName(rs.getString("doc_node_name"));
        eoaDocTreeVO.setParentId(rs.getString("parent_id"));
        eoaDocTreeVO.setIsend(rs.getString("isend"));
        eoaDocTreeVO.setDocName(rs.getString("doc_name"));
        eoaDocTreeVO.setIspublic(rs.getString("ispublic"));
        eoaDocTreeVO.setRemark(rs.getString("remark"));
        eoaDocTreeVO.setFindInfo(rs.getString("find_info"));
        eoaDocTreeVO.setAddInfo(rs.getString("add_info"));
        eoaDocTreeVO.setModInfo(rs.getString("mod_info"));
        eoaDocTreeVO.setDelInfo(rs.getString("del_info"));
        result.add(eoaDocTreeVO);
```

```
        }
    }
    catch(SQLException e)
    {
        Debug.println(e.getMessage());
        throw new DaoException("Error executing SQL SELECT doc_node_id ,
        region_id,doc_node_code,doc_node_name,parent_id,isend,doc_name ,
        ispublic,remark,find_info,add_info,mod_info,del_info FROM eoa_doc_tree
        WHERE 1=1 AND doc_node_name=? ORDER BY 1 " + e.getMessage());
    }
    finally
    {
        DBUtil.closeResultSet(rs);
        DBUtil.closeStatement(statement);
    }
    Debug.println( this.getClass().getName() + " Select end");
    return result ;
}

public java.util.Collection findByNodeCode(Connection transConn,
    java.lang.String docNodeCode ) throws DaoException//通过节点代码进行查询
{
    Debug.println( this.getClass().getName() + " Select begin");
    Connection conn = null;
    PreparedStatement statement = null;
    ResultSet rs = null;
    Vector result = new Vector();
    EoaDocTreeVO eoaDocTreeVO = null;
    try
    {
        conn = transConn;
        statement = conn.prepareStatement(" SELECT doc_node_id , region_id ,
        doc_node_code , doc_node_name , parent_id , isend , doc_name ,
        ispublic , remark , find_info , add_info , mod_info , del_info FROM
        eoa_doc_tree WHERE 1=1  AND doc_node_code = ?  ORDER BY 1 ");
        //条件
        statement.setString(1, DaoUtil.NullToStr(docNodeCode));
        rs = statement.executeQuery();
        while (rs.next())
        {
          eoaDocTreeVO = new EoaDocTreeVO();
          eoaDocTreeVO.setDocNodeId(rs.getString("doc_node_id"));
          eoaDocTreeVO.setRegionId(rs.getString("region_id"));
          eoaDocTreeVO.setDocNodeCode(rs.getString("doc_node_code"));
          eoaDocTreeVO.setDocNodeName(rs.getString("doc_node_name"));
          eoaDocTreeVO.setParentId(rs.getString("parent_id"));
          eoaDocTreeVO.setIsend(rs.getString("isend"));
          eoaDocTreeVO.setDocName(rs.getString("doc_name"));
          eoaDocTreeVO.setIspublic(rs.getString("ispublic"));
```

```
        eoaDocTreeVO.setRemark(rs.getString("remark"));
        eoaDocTreeVO.setFindInfo(rs.getString("find_info"));
        eoaDocTreeVO.setAddInfo(rs.getString("add_info"));
        eoaDocTreeVO.setModInfo(rs.getString("mod_info"));
        eoaDocTreeVO.setDelInfo(rs.getString("del_info"));
        result.add(eoaDocTreeVO);
      }
    }
    catch(SQLException e)
    {
      Debug.println(e.getMessage());
      throw new  DaoException("Error executing SQL  SELECT doc_node_id ,
      region_id,doc_node_code,doc_node_name,parent_id,isend,doc_name,
      ispublic , remark , find_info , add_info , mod_info , del_info FROM
      eoa_doc_tree WHERE 1=1 AND doc_node_code=? ORDER BY 1"
      + e.getMessage());
    }
    finally
    {
      DBUtil.closeResultSet(rs);
      DBUtil.closeStatement(statement);
    }
    Debug.println( this.getClass().getName() + " Select end");
    return result ;
  }
  public java.util.Collection findByParentId(Connection transConn,java.
    lang.String
  parentId ) throws DaoException //通过上级 ID 进行查询
  {
    Debug.println( this.getClass().getName() + " Select begin");
    Connection conn = null;
    PreparedStatement statement = null;
    ResultSet rs = null;
    Vector result = new Vector();
    EoaDocTreeVO eoaDocTreeVO = null;
    try
    {
      conn = transConn;
      statement = conn.prepareStatement(" SELECT doc_node_id , region_id ,
      doc_node_code , doc_node_name , parent_id , isend , doc_name ,
      ispublic , remark , find_info , add_info , mod_info , del_info FROM
      eoa_doc_tree WHERE 1=1  AND parent_id = ?  ORDER BY 1 ");
      //条件
      statement.setString(1, DaoUtil.NullToStr(parentId));
      rs = statement.executeQuery();
      while (rs.next())
      {
        eoaDocTreeVO = new EoaDocTreeVO();
        eoaDocTreeVO.setDocNodeId(rs.getString("doc_node_id"));
```

```
            eoaDocTreeVO.setRegionId(rs.getString("region_id"));
            eoaDocTreeVO.setDocNodeCode(rs.getString("doc_node_code"));
            eoaDocTreeVO.setDocNodeName(rs.getString("doc_node_name"));
            eoaDocTreeVO.setParentId(rs.getString("parent_id"));
            eoaDocTreeVO.setIsend(rs.getString("isend"));
            eoaDocTreeVO.setDocName(rs.getString("doc_name"));
            eoaDocTreeVO.setIspublic(rs.getString("ispublic"));
            eoaDocTreeVO.setRemark(rs.getString("remark"));
            eoaDocTreeVO.setFindInfo(rs.getString("find_info"));
            eoaDocTreeVO.setAddInfo(rs.getString("add_info"));
            eoaDocTreeVO.setModInfo(rs.getString("mod_info"));
            eoaDocTreeVO.setDelInfo(rs.getString("del_info"));
            result.add(eoaDocTreeVO);
          }
      }
      catch(SQLException e)
      {
         Debug.println(e.getMessage());
         throw new  DaoException("Error executing SQL  SELECT doc_node_id ,
         region_id , doc_node_code , doc_node_name , parent_id , isend ,
         doc_name,ispublic,remark,find_info,add_info,mod_info,del_info
          FROM eoa_doc_tree WHERE 1=1 AND parent_id=? ORDER BY 1"
          +e.getMessage());
      }
      finally
      {
         DBUtil.closeResultSet(rs);
         DBUtil.closeStatement(statement);
      }
      Debug.println( this.getClass().getName() + " Select end");
      return result ;
   }
   public java.util.Collection findByIsEnd(Connection transConn, java.lang.
String isend ) throws DaoException  //通过是否为最后节点进行查询
   {
      Debug.println( this.getClass().getName() + " Select begin");
      Connection conn = null;
      PreparedStatement statement = null;
      ResultSet rs = null;
      Vector result = new Vector();
      EoaDocTreeVO eoaDocTreeVO = null;
      try
      {
         conn = transConn;
         statement = conn.prepareStatement(" SELECT doc_node_id , region_id ,
         doc_node_code , doc_node_name , parent_id , isend , doc_name ,
         ispublic , remark , find_info , add_info , mod_info , del_info FROM
         eoa_doc_tree WHERE 1=1  AND isend = ?  ORDER BY 1 ");
         //条件
```

```
        statement.setString(1, DaoUtil.NullToStr(isend));
        rs = statement.executeQuery();
        while (rs.next())
        {
          eoaDocTreeVO = new EoaDocTreeVO();
          eoaDocTreeVO.setDocNodeId(rs.getString("doc_node_id"));
          eoaDocTreeVO.setRegionId(rs.getString("region_id"));
          eoaDocTreeVO.setDocNodeCode(rs.getString("doc_node_code"));
          eoaDocTreeVO.setDocNodeName(rs.getString("doc_node_name"));
          eoaDocTreeVO.setParentId(rs.getString("parent_id"));
          eoaDocTreeVO.setIsend(rs.getString("isend"));
          eoaDocTreeVO.setDocName(rs.getString("doc_name"));
          eoaDocTreeVO.setIspublic(rs.getString("ispublic"));
          eoaDocTreeVO.setRemark(rs.getString("remark"));
          eoaDocTreeVO.setFindInfo(rs.getString("find_info"));
          eoaDocTreeVO.setAddInfo(rs.getString("add_info"));
          eoaDocTreeVO.setModInfo(rs.getString("mod_info"));
          eoaDocTreeVO.setDelInfo(rs.getString("del_info"));
          result.add(eoaDocTreeVO);
        }
      }
      catch(SQLException e)
      {
        Debug.println(e.getMessage());
        throw new  DaoException("Error executing SQL  SELECT doc_node_id ,
        region_id , doc_node_code , doc_node_name , parent_id , isend ,
        doc_name , ispublic , remark , find_info , add_info , mod_info , del_info
        FROM  eoa_doc_tree WHERE 1=1  AND isend = ?
        ORDER BY 1 " + e.getMessage());
      }
      finally
      {
        DBUtil.closeResultSet(rs);
        DBUtil.closeStatement(statement);
      }
      Debug.println( this.getClass().getName() + " Select end");
      return result ;
    }
}
```

❑ VO（数值对象）映射

本系统中 VO 映射类如表 10-7 所示。

表 10-7　VO 映射类表

VO 映射类	描　　述
EoaDeptVO	部门 VO 映射类
EoaDocTreeVO	文件/目录 VO 映射类

续表

VO 映射类	描　　述
EoaTreeVO	树形 VO 映射类
PUserVO	用户 VO 映射类

下面以文件/目录信息表为例进行设计说明。文件/目录 VO 映射（EoaDocTreeVO）代码设计如下：

```
package com.dfkj.eoa.vo;
public class EoaDocTreeVO implements java.io.Serializable
{
  public EoaDocTreeVO()  {
  }
  private String docNodeId = "";
  private String regionId = "";
  private String docNodeCode = "";
  private String docNodeName = "";
  private String parentId = "";
  private String isend = "";
  private String docName = "";
  private String ispublic = "";
  private String remark = "";
  private String findInfo = "";
  private String addInfo = "";
  private String modInfo = "";
  private String delInfo = "";
  public void setDocNodeId(String docNodeId)       //设置文件节点 ID
  {
    this.docNodeId = docNodeId;
  }
  public String getDocNodeId()                     //获取文件节点 ID
  {
    return docNodeId;
  }
  public void setRegionId(String regionId)         //设置区域 ID
  {
    this.regionId = regionId;
  }
  public String getRegionId()                      //获取区域 ID
  {
    return regionId;
  }
  public void setDocNodeCode(String docNodeCode)   //设置文件节点代码
  {
    this.docNodeCode = docNodeCode;
  }
  public String getDocNodeCode()                   //获取文件节点代码
  {
    return docNodeCode;
```

```
}
public void setDocNodeName(String docNodeName)    //设置文件节点名称
{
   this.docNodeName = docNodeName;
}
public String getDocNodeName()                    //获取文件节点名称
{
   return docNodeName;
}
public void setParentId(String parentId)          //设置上级目录 ID
{
   this.parentId = parentId;
}
public String getParentId()                       //获取上级目录 ID
{
   return parentId;
}
public void setIsend(String isend)                //设置是否为最后节点
{
   this.isend = isend;
}
public String getIsend()                          //获取是否为最后节点
{
   return isend;
}
public void setDocName(String docName)            //设置文件/目录名称
{
   this.docName = docName;
}
public String getDocName()                        //获取文件/目录名称
{
   return docName;
}
public void setIspublic(String ispublic)          //设置目录是否公开
{
   this.ispublic = ispublic;
}
public String getIspublic()                       //获取目录是否公开
{
   return ispublic;
}
public void setRemark(String remark)              //设置备注
{
   this.remark = remark;
}
public String getRemark()                         //获取备注
{
   return remark;
}
```

```
    public void setFindInfo(String findInfo)            //设置查询权限信息
    {
       this.findInfo = findInfo;
    }
    public String getFindInfo()                         //获取查询权限信息
    {
       return findInfo;
    }
    public void setAddInfo(String addInfo)              //设置增加权限信息
    {
       this.addInfo = addInfo;
    }
    public String getAddInfo()                          //获取增加权限信息
    {
       return addInfo;
    }
    public void setModInfo(String modInfo)              //设置修改权限信息
    {
       this.modInfo = modInfo;
    }
    public String getModInfo()                          //获取修改权限信息
    {
       return modInfo;
    }
    public void setDelInfo(String delInfo)              //设置删除权限信息
    {
       this.delInfo = delInfo;
    }
    public String getDelInfo()                          //获取删除权限信息
    {
       return delInfo;
    }
}
```

❑ 服务层设计

在本系统中采用 Struts 来构造系统的框架，因此这里所有的业务逻辑都是通过 Actions 和 Business 组合起来的。表现层 JSP 通过 Actions 请求类向 Business 业务逻辑类发出请求，Business 通过 DAO 及 VO 与数据库进行交互，并进行一系列的处理工作。各个部分之间的具体关系参考图 10-8。

➢ Actions 请求类

本系统中 Actions 请求类如表 10-8 所示。

表 10-8　Actions 请求类

Actions 请求类	描　　述
FindEoaDocTreeByParentIdAction	通过上级目录 ID 查询文件请求
FindEoaDocTreeByDocNodeIdAction	通过目录节点 ID 查询文件请求
FindEoaDocTreeByNodeNameAction	通过节点名称查询文件请求

续表

Actions 请求类	描　　述
DeleteEoaDocTreeAction	删除文件请求
FindEoaDocTreeLeafByParentIdAction	通过上级 ID 查询文件子节点请求
AddEoaDocTreeAction	增加文件请求
FindEoaDocTreeDirectoryByParentIdAction	通过上级 ID 查询目录请求
FindEoaDocTreeDirectoryByAllAction	通过所有条件查询目录请求
FindEoaDocTreeDirectoryByNodeNameAction	通过节点名称查询目录请求
UpdateEoaDocTreeAction	修改文件请求
FileUpload	文件上传请求

- Business 业务逻辑类

Business 业务逻辑的实现一般需要 3 个部分：

- Business 工厂。
- Business 接口。
- Business 接口的实现。

3 个部分之间的关系如图 10-13 所示。

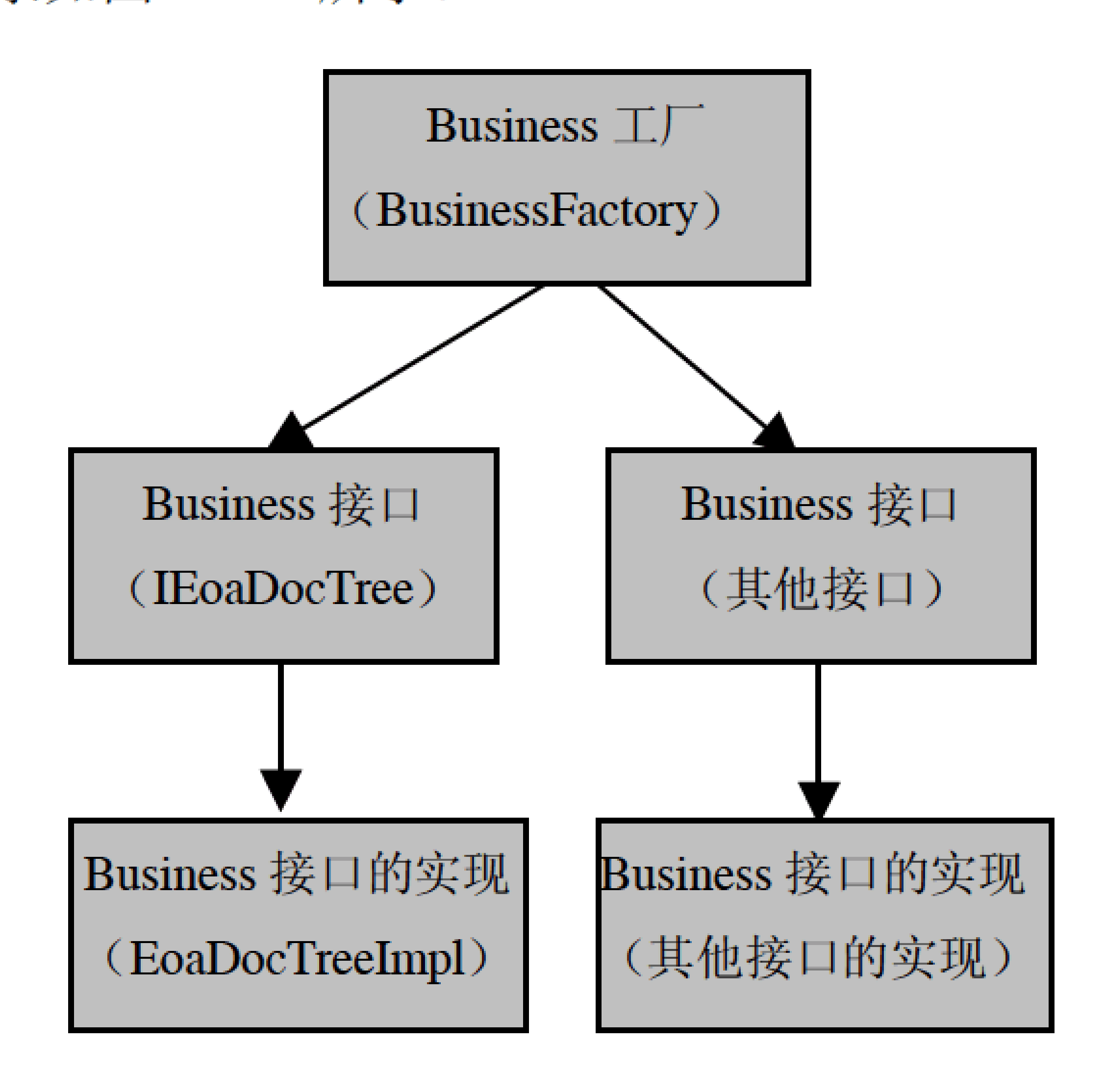


图 10-13　Business 业务逻辑各部分之间关系图

本系统中，Business 工厂类（BusinessFactory）代码设计如下：

```
package com.dfkj.eoa.business;
public class BusinessFactory {  /* 对各业务逻辑提供接口工厂 */
    private BusinessFactory() {
    }
    public static BusinessFactory newInstance() {
        return new BusinessFactory();
    }
    public IEoaDocTree buildEoaDocTreeImpl() {
        return new EoaDocTreeImpl();
```

```
    }
}
```

本系统中 Business 接口类设计如表 10-9 所示。

表 10-9　Business 接口类表

Business 接口类	描　　述
IEoaDocTree	文件/目录管理 Business 接口类

IEoaDocTree 接口类代码设计如下：

```
package com.dfkj.eoa.business;
import java.sql.Connection;
import java.util.Collection;
import com.dfkj.exception.*;
import com.dfkj.eoa.vo.EoaDocTreeVO;
public interface IEoaDocTree {
    public void addEoaDocTree(Connection transConn,EoaDocTreeVO
      eoaDocTreeVO) throws DaoException;                  //增加文件/目录
    public void updateEoaDocTree(Connection transConn,EoaDocTreeVO
      eoaDocTreeVO) throws DaoException;                  //修改文件/目录
    public void deleteEoaDocTree(Connection transConn,java.lang.String
      docNodeId) throws DaoException;                     //删除文件/目录
    public EoaDocTreeVO findByPrimaryKey(Connection     //通过主键查询
      transConn,java.lang.String  docNodeId) throws DaoException;
    public java.util.Collection findAll(Connection transConn)
      throws DaoException;                                //查询所有文件/目录
    public java.util.Collection findByCondition(Connection//通过给定条件查询
      transConn,java.util.Properties condition) throws DaoException;
    public java.util.Collection findByNodeName(Connection transConn,
      java.lang.String docNodeName ) throws DaoException; //通过节点名称查询
    public java.util.Collection findByNodeCode(Connection transConn,
      java.lang.String docNodeCode ) throws DaoException; //通过节点代码查询
    public java.util.Collection findByParentId(Connection transConn,
      java.lang.String parentId ) throws DaoException;  //通过上级目录 ID 查询
    public java.util.Collection findByIsEnd(Connection transConn,
      java.lang.String isend ) throws DaoException;    //通过是否为最后节点查询
}
```

本系统中 Business 接口的实现类设计如表 10-10 所示。

表 10-10　Business 接口的实现类表

Business 接口的实现类	描　　述
EoaDocTreeImpl	文件/目录管理 Business 接口的实现类

EoaDocTreeImpl 类的程序设计如下：

```
package com.dfkj.eoa.business;
import java.sql.Connection;
import java.util.Collection;
```

```
import java.util.Properties;
import com.dfkj.eoa.vo.EoaDocTreeVO;
import com.dfkj.exception.DaoException;
import java.util.Vector;
import com.dfkj.eoa.dao.*;

 public class EoaDocTreeImpl implements IEoaDocTree {
   private IEoaDocTreeDAO iEoaDocTreeDAO;
   public EoaDocTreeImpl() {
      iEoaDocTreeDAO = DAOFactory.newInstance().buildEoaDocTreeDAO();
   }
     public void addEoaDocTree(Connection conn, EoaDocTreeVO
       eoaDocTreeVO)  throws DaoException {          //增加文件/目录
         iEoaDocTreeDAO.insert(conn, eoaDocTreeVO);
   }
     public void deleteEoaDocTree(Connection conn, String docNodeId)
       throws DaoException {                         //删除文件/目录
         iEoaDocTreeDAO.delete(conn, docNodeId);
   }
     public void updateEoaDocTree(Connection conn, EoaDocTreeVO
       eoaDocTreeVO)  throws DaoException {          //修改文件/目录
         iEoaDocTreeDAO.update(conn, eoaDocTreeVO);
   }
     public EoaDocTreeVO findByPrimaryKey(Connection conn, String docNodeId)
       throws DaoException {                         //通过主键查询
         return iEoaDocTreeDAO.findByPrimaryKey(conn, docNodeId);
   }
     public Collection findAll(Connection transConn) throws DaoException{
         return iEoaDocTreeDAO.findAll(transConn); //查询所有文件/目录
   }
    public Collection findByCondition(Connection transConn,Properties condition)
       throws DaoException {                         //通过给定条件查询
         return iEoaDocTreeDAO.findByCondition(transConn, condition);
   }
     public Collection findByNodeName(Connection transConn, String
       docNodeName ) throws DaoException{            //通过节点名称查询
         return iEoaDocTreeDAO.findByNodeName(transConn, docNodeName);
   }
     public Collection findByNodeCode(Connection transConn, String
       docNodeCode ) throws DaoException{            //通过节点代码查询
         return iEoaDocTreeDAO.findByNodeCode(transConn, docNodeCode);
   }
    public Collection findByParentId(Connection transConn, String parentId )
      throws DaoException{                           //通过上级目录 ID 查询
        return iEoaDocTreeDAO.findByParentId(transConn, parentId);
   }
     public Collection findByIsEnd(Connection transConn, String isend )
       throws DaoException{                          //通过是否为最后节点查询
         return iEoaDocTreeDAO.findByIsEnd(transConn, isend);
```

```
    }
}
```

10.5　运行与调试本章的案例

将源代码中提供的本章应用程序（整个目录 FILE）复制到本机所安装的\Apache Software Foundation\Tomcat 5.5\webapps 路径下。

由于本章提供的案例是在 Oracle 8.17 数据库下开发的，所以读者可以在 Oracle 8.17 及以上版本的数据库下创建用户 file1，其密码为 file1，然后将 file.dmp 直接导入到数据库即可。

另外，读者可以将程序进行简单修改（主要是 DAO），将数据库迁移到 MySQL 5.1。

在 Tomcat 5.5 中要进行调试，跟踪调试信息，可以直接运行 Tomcat 5.5 中的 tomcat5.exe 程序。

❑　配置环境变量

在“我的电脑”上单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，弹出“系统特性”对话框，选择“高级”选项卡，然后单击“环境变量”按钮，即可编辑系统的环境变量。

❑　设置 Server.xml 文件

Tomcat 主目录/conf 下的 server.xml 文件是对 Web 服务器的配置。找到以下代码：

```
<Connector
       URIEncoding="GBK"
       port="8080"
       redirectPort="8443"
       minSpareThreads="25"
       connectionTimeout="20000"
       uRIEncoding="GBK"
       maxSpareThreads="75"
       maxThreads="150"
       maxHttpHeaderSize="8192">
</Connector>
```

可以将 8080 端口改为喜欢使用的端口，如常见的 80（只要不冲突），以后即可利用该端口访问系统 http://localhost:80/FILE。

如果本章案例应用程序（\FILE 整个目录）不放在\Apache Software Foundation\Tomcat 5.5\webapps 路径下，例如放在开发路径 d:\myeclipseproject 下，那么在 server.xml 文件中找到以下代码：

```
<Host
      appBase="webapps"
      name="localhost">
</Host>
```

在其间添加一个<Context>元素：

```
<Context path="/FILE"  reloadable="true"
   docBase=" d:\myeclipseproject\FILE"
   workDir=" d:\myeclipseproject\FILE\work" >
</Context>
```

在浏览器中打开 http://localhost:80/FILE 时就会转向 d:\myeclipseproject\FILE 下的应用程序。

❑ 设置 web.xml 文件

刚开始调试时，可以对\Tomcat5.5\webapps\FILE\WEB-INF 下的 web.xml 中的代码进行修改，如下所示：

```
<filter>
   <!-- 权限过滤器定义 -->
        <filter-name>com.dfkj.web.PowerFilter</filter-name>
        <filter-class>com.dfkj.web.PowerFilter</filter-class>
        <init-param>
            <param-name>flag</param-name>
            <param-value>off</param-value>  <!--off 表示权限有效，on 无效 -->
        </init-param>
    </filter>
```

将 off 改为 on，这样权限控制将不起作用，输入用户名和密码，可以进入任何模块，主要是为了调试方便。

10.6　小　　结

本章以“文件管理系统”的开发与实现为主线，从需求分析、系统架构设计、数据库设计、系统包设计、系统关键技术、系统主要模块具体实现等方面逐步介绍，较完整地讲解了该系统的分析、设计与编程实现过程，可综合之前所学的基础知识。

系统结合 MVC 模式、CSS 技术、分页技术等知识，给读者提供了一个真实工程的学习与练习环境。

第 11 章　教务管理系统案例

本章将要实现的是学生课程及成绩管理系统，系统方便了学生选课和查分，方便了教师的教学管理和学生成绩的录入，更为主要的是方便了学校的教务管理。

本系统将采用 MVC 三层架构的模式，在开发过程中使用到 Struts 和 Hibernate 处理页面逻辑和对象的持久化工作。本系统的开发并没有单纯地使用 JSP+Servlet 进行开发，而是结合了 Struts 和 Hibernate，这是为了使系统的结构更加清晰同时简化开发工作。

在系统开发前，读者应该有网页设计的基本知识，可以熟练地使用数据库（本章中使用 MySQL）和 Web 服务器（本章中使用 Tomcat），以及熟练掌握 Struts 框架和 Hibernate 的使用方法。

11.1　系统需求分析

11.1.1　需求概述

教务管理系统的设计目的，是要将学生选择的课程和学生成绩通过网络进行管理。为学生、教师和教务管理人员提供便利。系统的用户共有 3 种类型，分别为系统管理员、学生及教师，系统对于一个用户只允许以一种身份登录。系统管理员登录系统后可以对系统进行管理，其主要操作是维护学生、教师、课程和班级的基本信息。学生登录后的主要操作是选课和个人信息维护。教师登录后的主要操作是选择学生并为学生登录成绩。

将系统需求加以总结，得出系统需求列表如下：

- 系统可以运行在 Windows 操作系统平台下，并通过友好的用户界面。
- 系统用户类型为管理员、教师、学生。
- 系统对于一个用户只允许以一种身份登录。
- 只有管理员可以维护学生、教师、课程、班级的基本信息。
- 学生可以选课并维护自己的个人信息。
- 教师可以选择上课的学生并为学生登录成绩。

11.1.2　系统功能描述

通过前面的分析已经明确系统用户共有以下 3 类。

- 管理员：管理学生、教师、课程和班级信息。
- 学生：选课、查看成绩、修改个人信息。

- ❑ 教师：选择学生、登录成绩。

不同的用户可以通过系统进行不同的操作，每一操作都是一个功能的体现，下面给出系统需要实现的具体功能。

1. 用户登录功能

不同用户登录系统时首先选择对应的用户类型，然后输入用户名及密码登录系统。系统的管理员由系统内部设定，学生和教师由管理员添加（或者与教务管理系统中其他子系统共用数据）。用户登录系统的页面如图 11-1 所示。

2. 管理员登录后选择功能

管理员登录后，会得到欢迎信息表示登录成功；如果登录失败，则会有错误提示信息。管理员可以在如图 11-2 所示的页面中单击“学生”、“教师”、“课程”和“班级”这 4 个链接进入不同页面继续下一步的操作。



图 11-1　用户登录页面

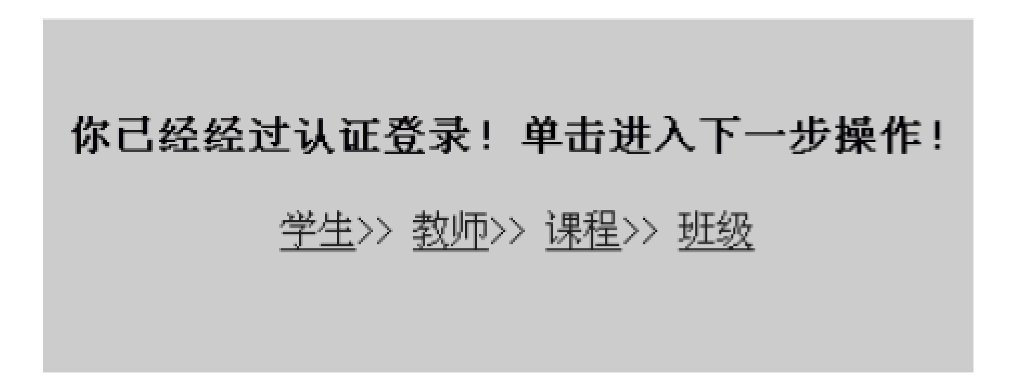


图 11-2　管理员管理首页

3. 管理员管理学生功能（查看、添加、编辑以及删除学生信息）

管理员管理学生包括查看、添加、编辑以及删除学生的信息。为了让读者更加清楚系统的功能，下面将系统功能与页面效果相结合，逐一加以描述。

- ❑ 管理员查看学生信息

当管理员进行学生信息管理操作时，即在图 11-2 所示页面中单击“学生”链接，将跳转到一个显示所有学生信息列表的页面，页面效果如图 11-3 所示。

- ❑ 管理员添加学生信息

在如图 11-3 所示的页面上，单击学生列表右上方的“新加学生”链接，将会跳转到如图 11-4 所示的增加学生页面，系统管理员可以通过这个页面向系统中添加学生信息。

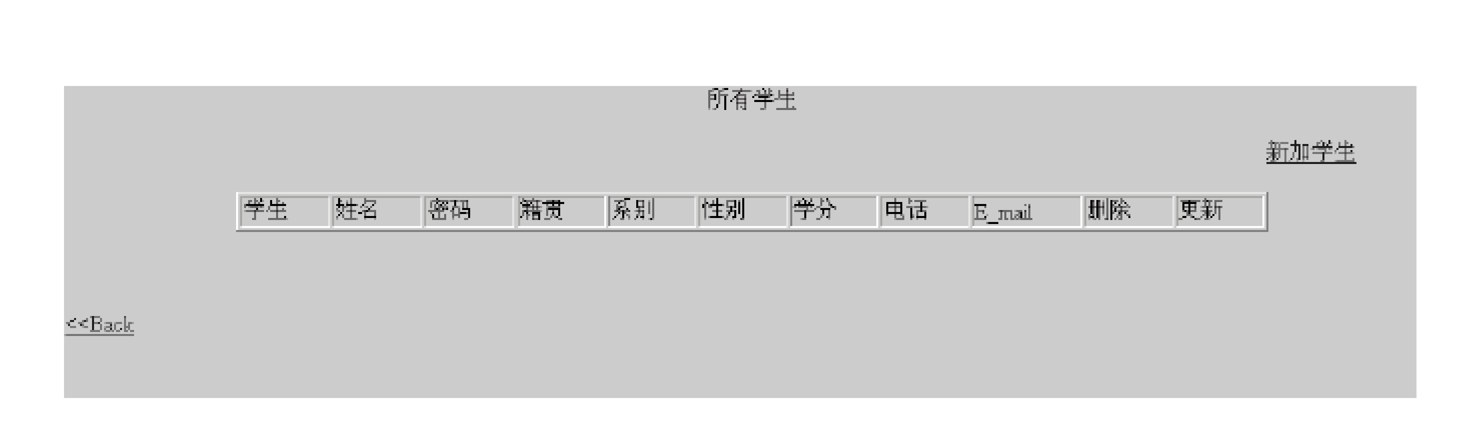


图 11-3　显示学生列表界面



图 11-4　添加学生信息

- ❑ 管理员编辑学生信息

在图 11-3 所示页面上单击学生列表中某条学生记录链接时，将会跳转到如图 11-5 所示的修改学生信息页面，系统管理员可以通过这个页面编辑、更新学生的信息。

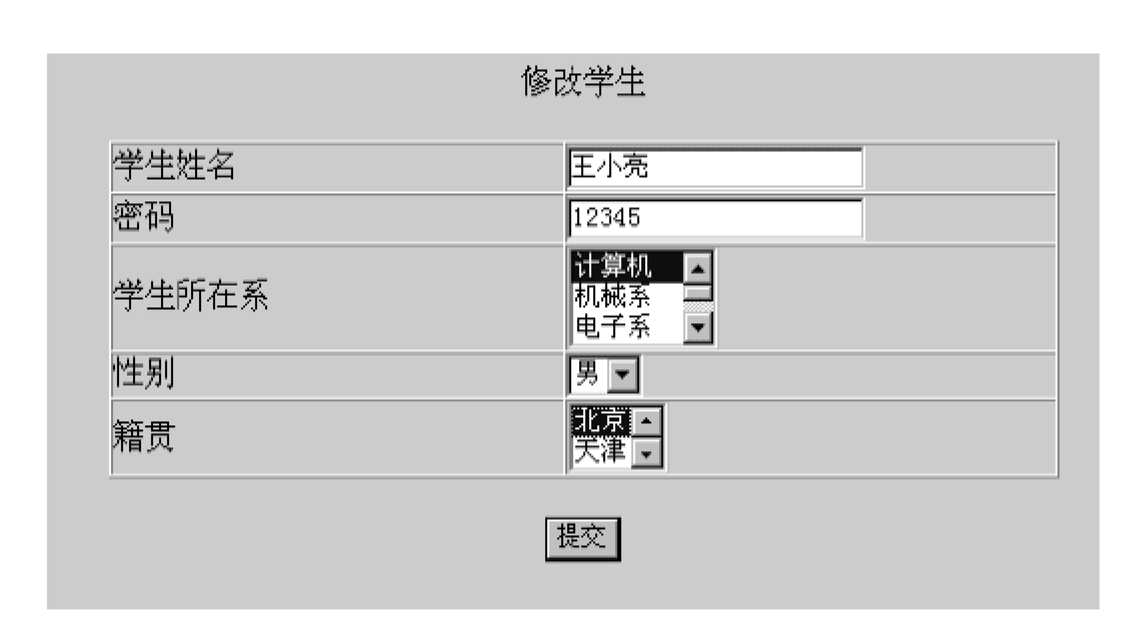


图 11-5　修改学生界面设计

- 管理员删除学生信息

在图 11-3 中每个学生记录的后面都有一个“删除”链接，单击它，系统将删除这位学生的信息。

4．管理员管理教师功能（查看、添加、编辑以及删除教师信息）

管理员管理课程包括查看、添加、编辑以及删除课程的信息。为了让读者更加清楚系统的功能，下面将系统功能结合页面效果逐一加以描述。

- 管理员查看课程信息

当管理员进行课程信息管理操作时，即在图 11-2 所示页面中单击“课程”链接，将跳转到一个显示所有课程信息列表的页面，页面效果如图 11-6 所示。

- 管理员添加课程信息

在图 11-6 所示页面上单击课程列表右上方的“新增课程”链接，将会跳转到如图 11-7 所示的新增课程信息页面，系统管理员可以通过这个页面向系统中添加课程信息。

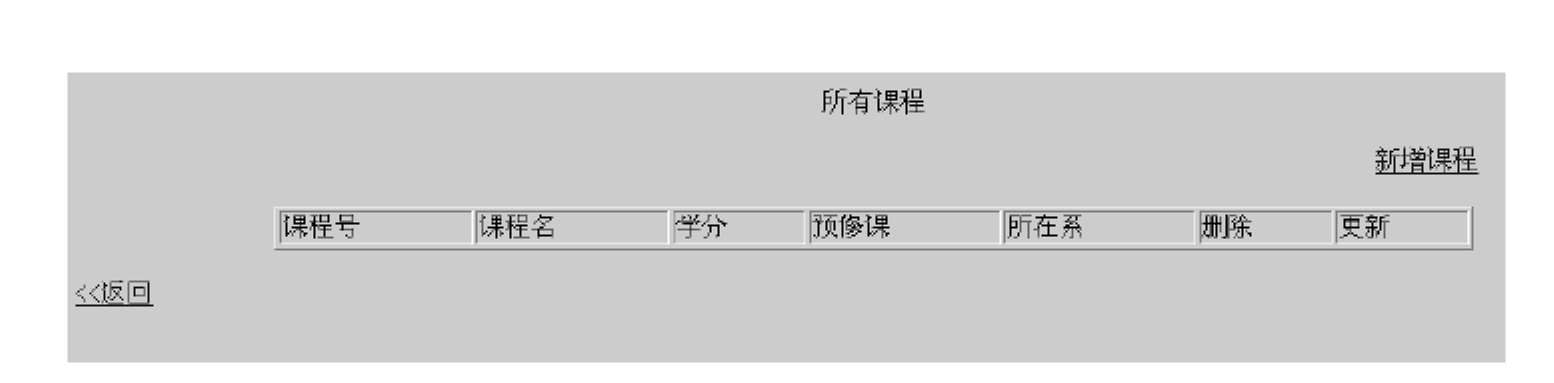


图 11-6　课程列表

图 11-7　添加新课程界面设计

- 管理员编辑课程信息

单击课程列中某条课程记录链接时，将会跳转到如图 11-8 所示的更新课程信息页面，系统管理员可以通过这个页面编辑、更新课程的信息。

- 管理员删除课程信息

在图 11-6 中每个课程记录的后面都有一个“删除”链接，单击它，删除这个课程的信息。

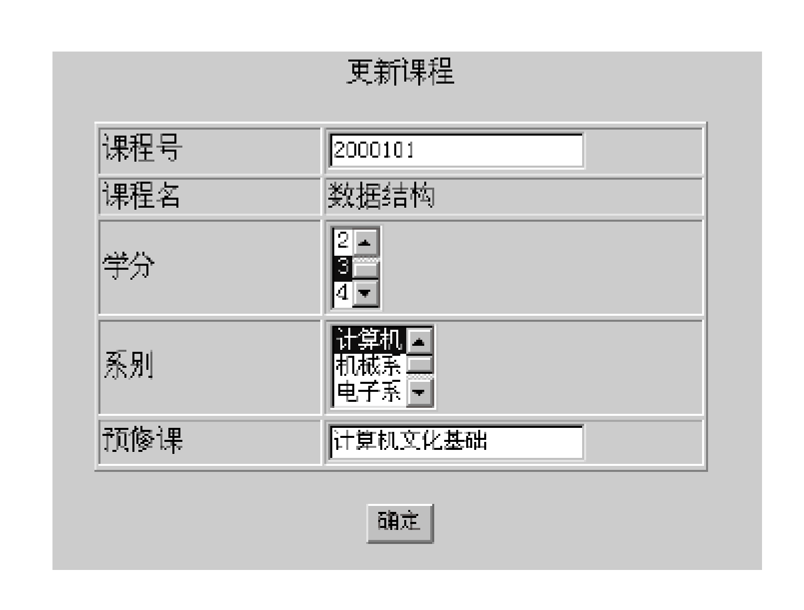


图 11-8　添加新课程界面设计

5. 管理员管理班级功能（查看、添加、编辑以及删除班级信息）

管理员管理班级包括查看、添加、编辑以及删除班级的信息。为了让读者更加清楚系统的功能，下面将系统功能与页面效果相结合，逐一加以描述。

- ❑ 管理员查看班级信息

当管理员进行班级信息管理操作时，即在图 11-2 所示页面中单击“班级”链接，将跳转到一个显示所有班级信息列表的页面，页面效果如图 11-9 所示。

- ❑ 管理员添加班级信息

在图 11-9 所示页面上单击班级列表右上方的“新增班级”链接，将会跳转到如图 11-10 所示的新增班级信息页面，系统管理员可以通过这个页面向系统中添加班级信息。

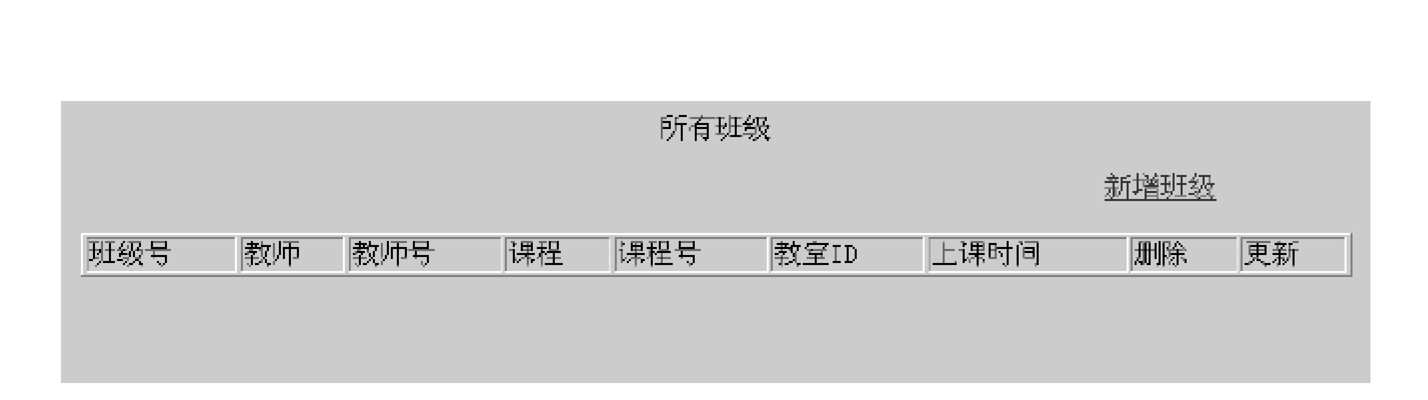


图 11-9　班级列表

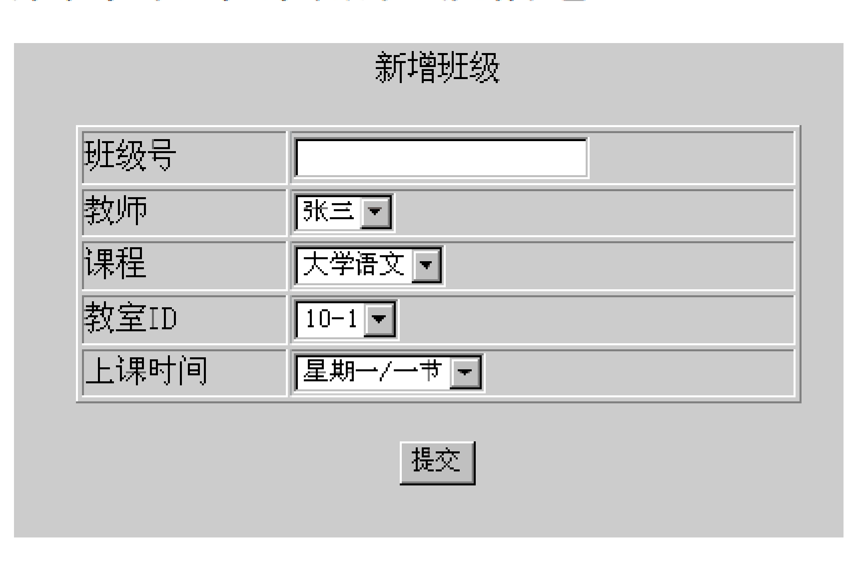


图 11-10　添加新班级界面设计

- ❑ 管理员编辑班级信息

在图 11-9 所示页面上单击班级列中某条班级记录链接时，将会跳转到如图 11-11 所示的更新班级信息页面，系统管理员可以通过这个页面编辑、更新班级的信息。

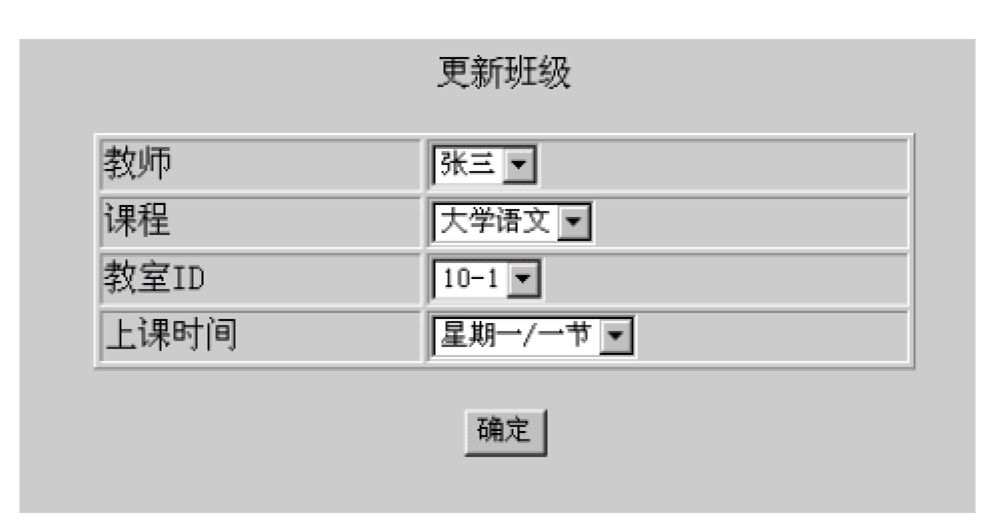


图 11-11　添加新班级界面设计

- ❑ 管理员删除班级信息

在图 11-9 中每个班级记录的后面都有一个“删除”链接，单击它，系统将删除这个班级的信息。

6. 学生用户登录后选择功能

学生登录本系统后，会得到欢迎信息表示登录成功；如果登录失败，则会有错误提示信息。学生可以在如图 11-12 所示的页面中单击“选修课程”、“查看学分”和“更改信息”这 3 个链接进入不同页面继续下一步的操作。

7. 学生选修课程功能

学生登录成功后，通过单击图 11-12 所示页面中的“选修课程”链接可以跳转到如

图 11-13 所示页面。这个页面显示了学生所有能够选择的课程列表，并在每条课程信息之后放置了一个“注册”链接，学生可以通过单击这个链接来选修这门课程。

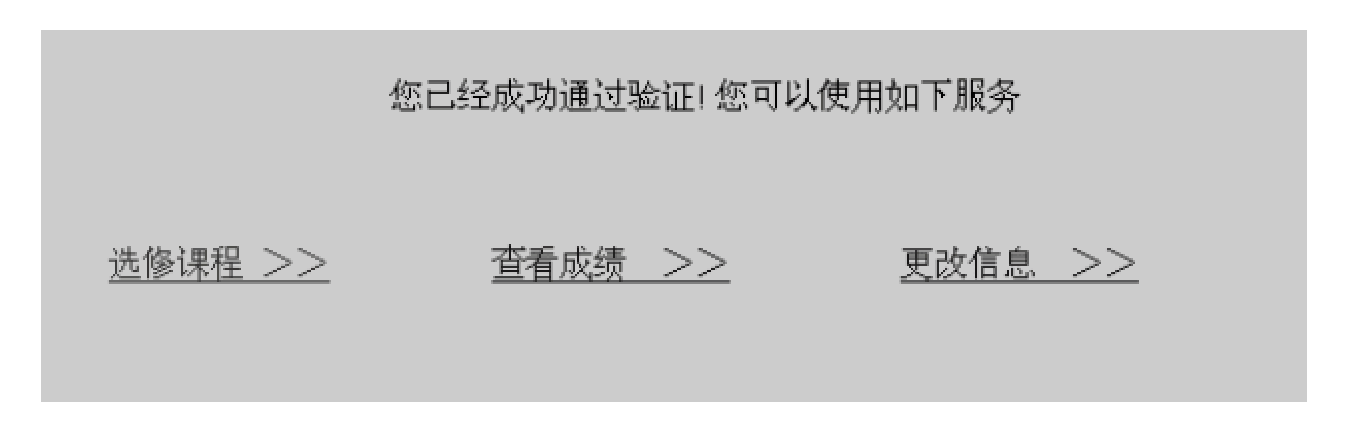
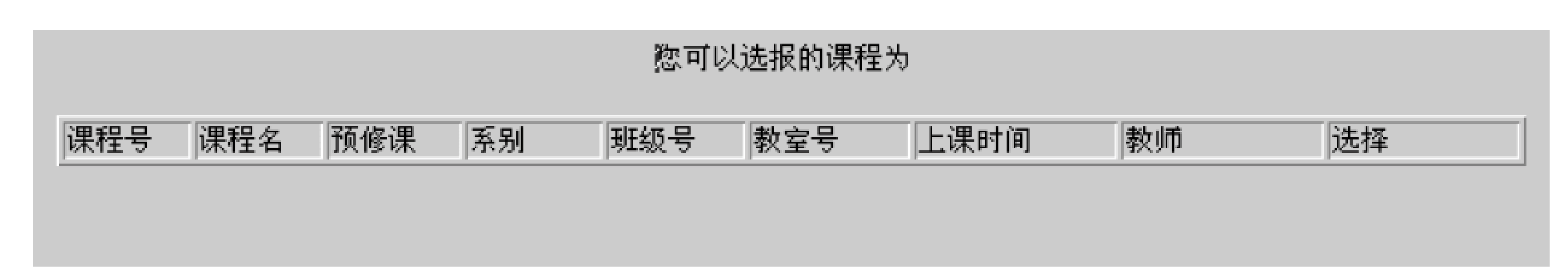


图 11-12　学生选择功能



您可以选报的课程为

课程号	课程名	预修课	系别	班级号	教室号	上课时间	教师	选择

图 11-13　学生选课

8．学生查看成绩功能

通过单击图 11-12 所示页面中的“查看成绩”链接可以跳转到如图 11-14 所示页面。这个页面显示了学生所有课程的成绩列表。

9．学生更新个人信息功能

单击“更改信息”链接可以进入到如图 11-15 所示页面。学生可以通过这个页面修改自己的一些信息。

您所有的成绩

课程名称	学分	成绩
大学语文	2	87分
高等数学	3	92分

图 11-14　学生查看成绩



图 11-15　学生修改信息

10．教师用户登录后选择功能

教师登录后，会得到欢迎信息表示登录成功；如果登录失败，则会有错误提示信息。教师可以在如图 11-16 所示的页面中单击“选择学生”和“公布成绩”这两个链接进入不同页面继续下一步的操作。

11．教师选择学生功能

通过单击图 11-16 所示页面中的“选择学生”链接可以跳转到如图 11-17 所示页面。这个页面显示了教师所教授的课程列表，并在每条课程信息之后放置了一个“选择”链接，教师可以通过单击这个链接来选择报名这门课程的学生。

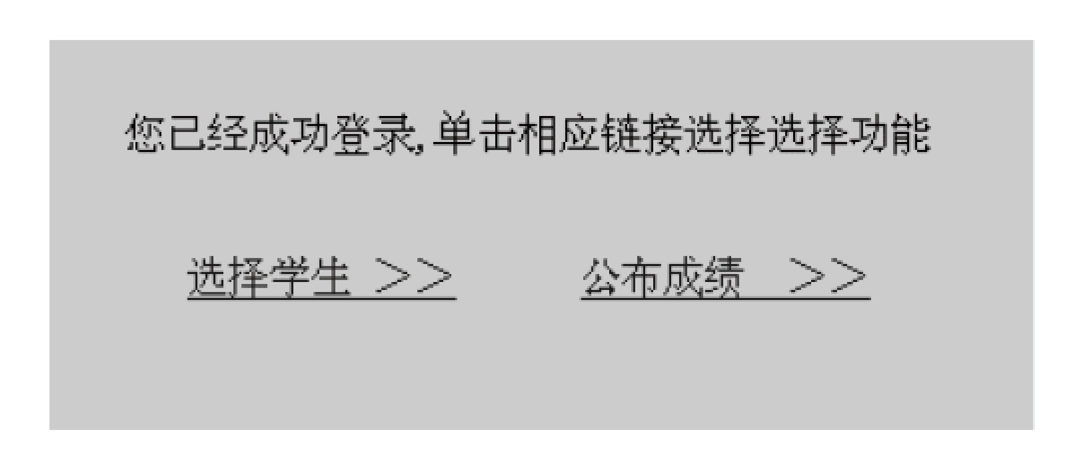


图 11-16　教师选择功能

教师教授课程

学生号	课程号	操作
001	200103	选择
002	200101	选择
003	200102	选择

图 11-17　教师教授课程列表

单击图 11-17 所示页面中的“选择”链接后，将跳转到如图 11-18 所示页面，在这个页面中列出了所有报名这门课程的学生信息。教师可以通过单击某条学生记录链接来选择这名学生。

12．教师登录成绩功能

单击“公布成绩”链接跳转到课程列表页面。这个页面显示了教师所教授的课程列表，并在每条课程信息之后放置了一个“选择”链接，教师可以通过单击这个链接跳转到学生列表页面，再通过单击某位学生信息后的链接，跳转到如图 11-19 所示的成绩录入页面（这个过程与“教师选择学生功能”的操作顺序类似）。教师用这个页面给学生打分。

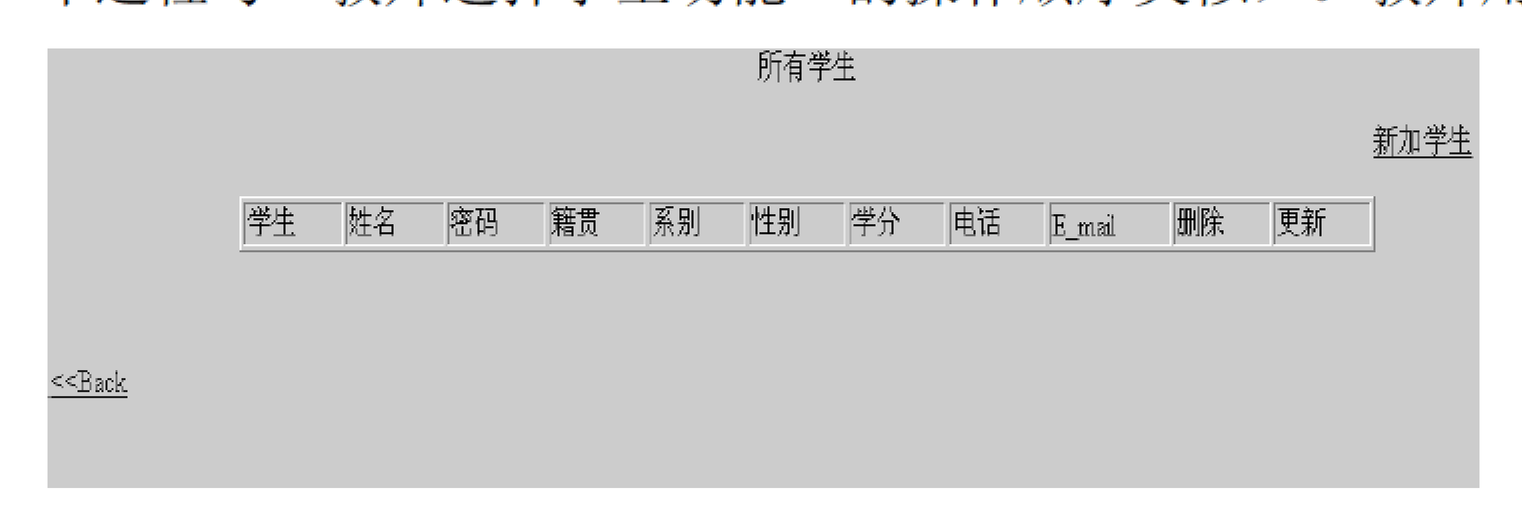


图 11-18　教师选择学生

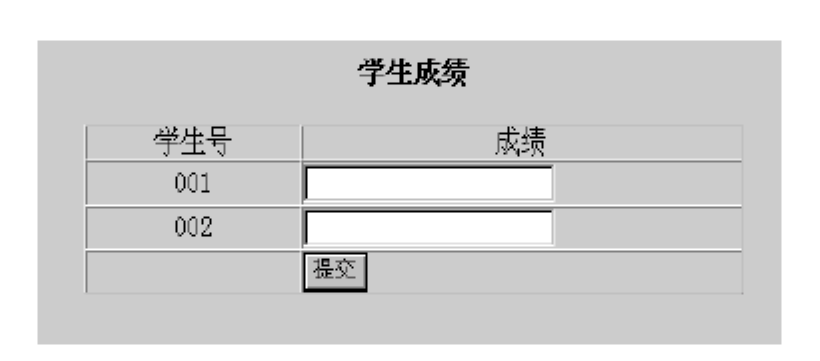


图 11-19　学生成绩录入

11.1.3　系统分析

模块分析和系统流程分析是描述系统需求的一个过程，需要将需求分析中的感性描述进行抽象，提取出要实现的功能，这是整个系统开发的一个关键过程。

1．系统功能模块划分

学生课程及成绩管理系统的需求分析，应该由开发人员和用户或者客户一起完成。这里的例子只用于帮助读者学习系统开发的过程及方法，所以对于将要开发实现的学生课程及成绩管理系统，实际上并没有真正的用户或客户，在开发过程中假定笔者就是系统的使用者，并由此提出具体需求。系统的模块结构如图 11-20 所示。

需求分析的第一步，是描述学生课程及成绩管理系统的功能，以此确定系统的功能需求。学生课程及成绩管理系统的角色是管理员、学生和教师，管理员对学生、教师、课程和班级信息进行维护，学生选择想要上的课程，查看所选的学分以及修改个人信息，教师决定上课的学生以及给学生学分。根据以上的用户操作需求，将系统划分为如下 3 大功能，并对其模块的划分和功能进行描述。

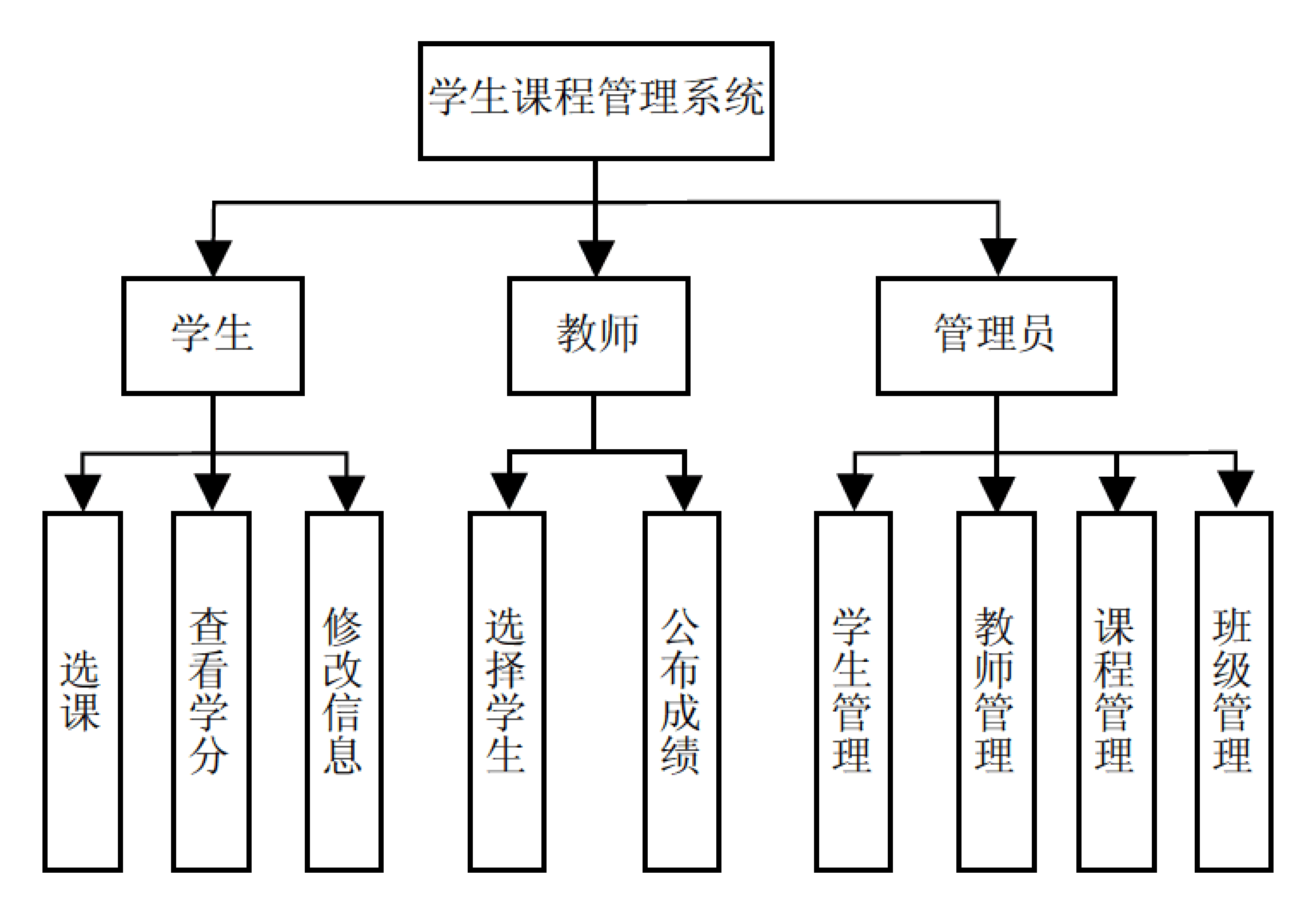


图 11-20　系统的模块结构

- ❑ 管理员功能
  - ➢ 登录：登录。
  - ➢ 学生管理：列表、增加、修改、删除。
  - ➢ 教师管理：列表、增加、修改、删除。
  - ➢ 课程管理：列表、增加、修改、删除。
  - ➢ 班级管理：列表、增加、修改、删除。
- ❑ 学生功能
  - ➢ 登录：登录。
  - ➢ 选课：选课。
  - ➢ 学分：查看。
  - ➢ 个人信息：修改。
- ❑ 教师功能
  - ➢ 登录：登录。
  - ➢ 选择学生：课程列表、学生列表、选择。
  - ➢ 公布成绩：课程列表、学生列表、成绩。

2．系统流程分析

本系统中的中心对象是学生和教师。根据以上的模块划分和功能分析可知，该系统的流程主要描述的是学生选择课程后，教师根据选课的学生决定选哪些学生，最后教师给学生学分。该系统的适用对象包括学生、教师和管理员，因此包括 3 个基本的流程。

图 11-21 描述的是管理员的操作流程：首先管理员要进行学生、教师、课程和班级数据的初始化，也就是说由管理员将学生和教师的信息加入到系统的数据库中。当然，也可以不通过系统而直接通过数据库将已有的学生和教师信息导入，这种做法与前一种做法的效果相同。之后学生和教师就可以登录并使用系统。在系统使用过程中，管理员进行管理工作。

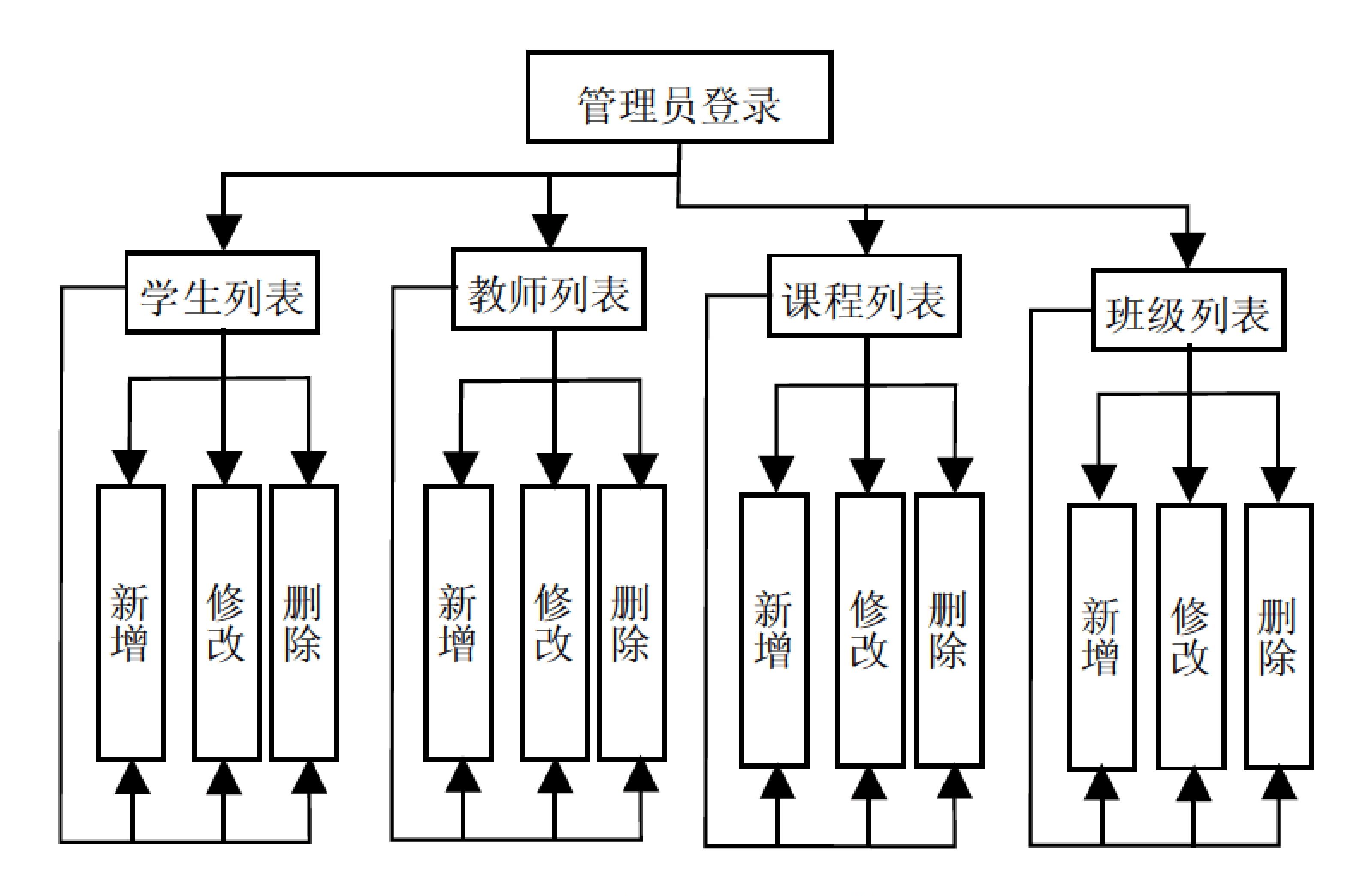


图 11-21　系统流程图—管理员

图 11-22 描述的是学生的操作流程：学生根据学生号和密码登录系统，初始的密码由管理员提供。学生登录系统后，可以修改个人信息、选课和查看学分。

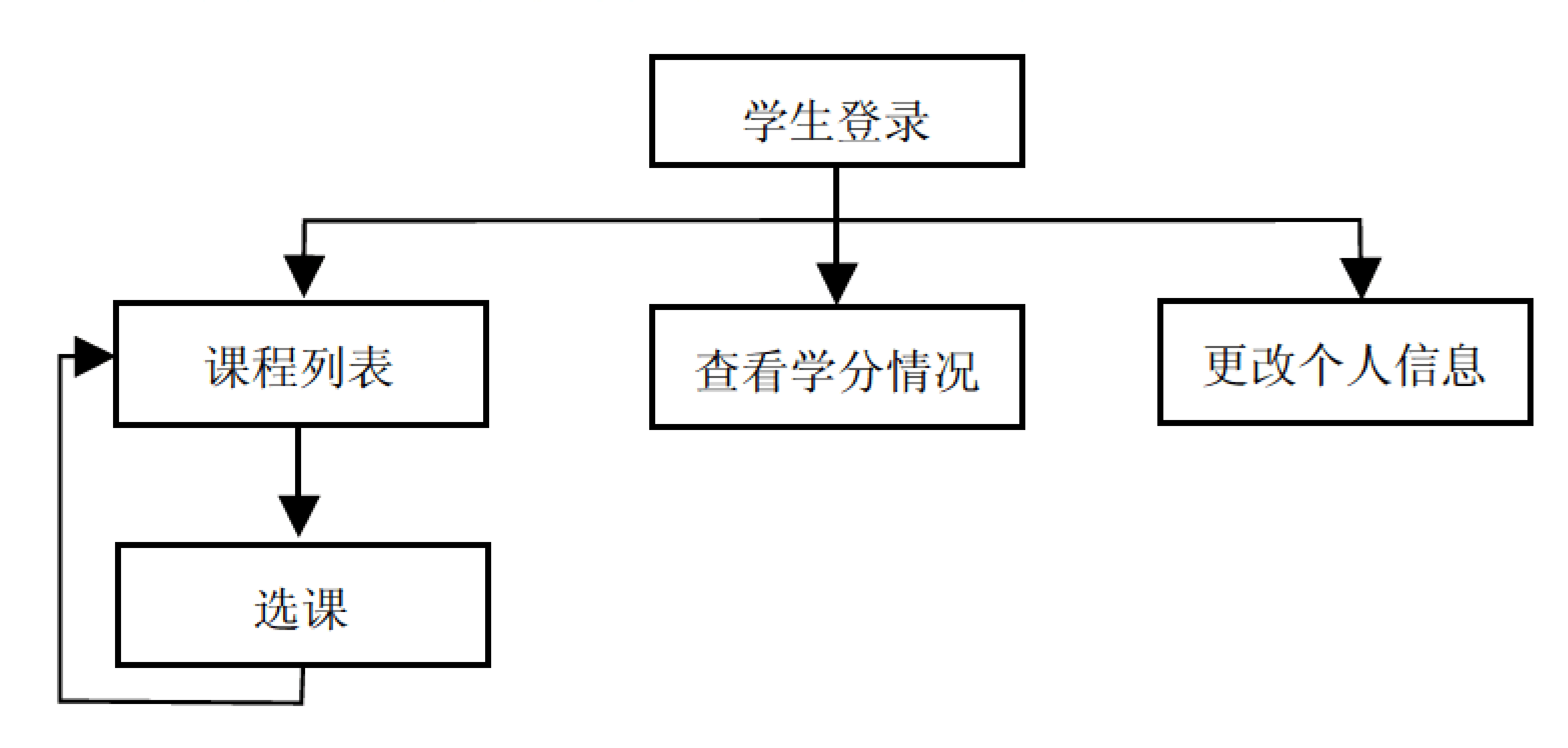


图 11-22　系统流程图—学生

图 11-23 描述的是教师的操作流程：教师根据教室号和密码登录系统，初始的密码由管理员提供。教师登录系统后，选择学习本课程的学生和给学生学分。

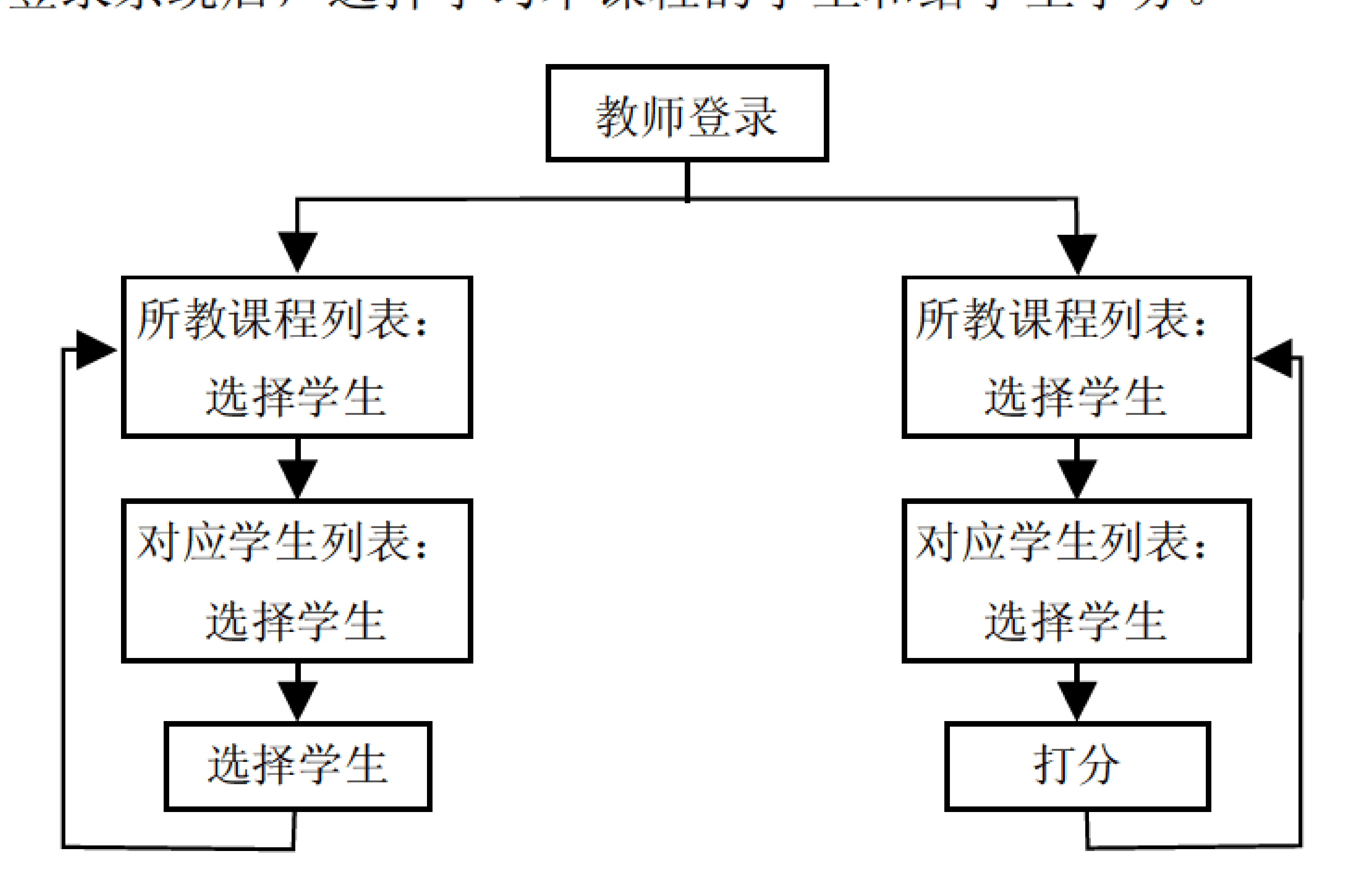


图 11-23　系统流程图—教师

11.2　系统总体架构

本系统采用 J2EE 的三层结构，分为表现层、业务逻辑层和数据服务层。三层体系将业务规则、数据访问等工作放到中间层处理，客户端不直接与数据库交互，而是通过控制器与中间层建立连接，再由中间层与数据库交互。

系统总体架构如图 11-24 所示。

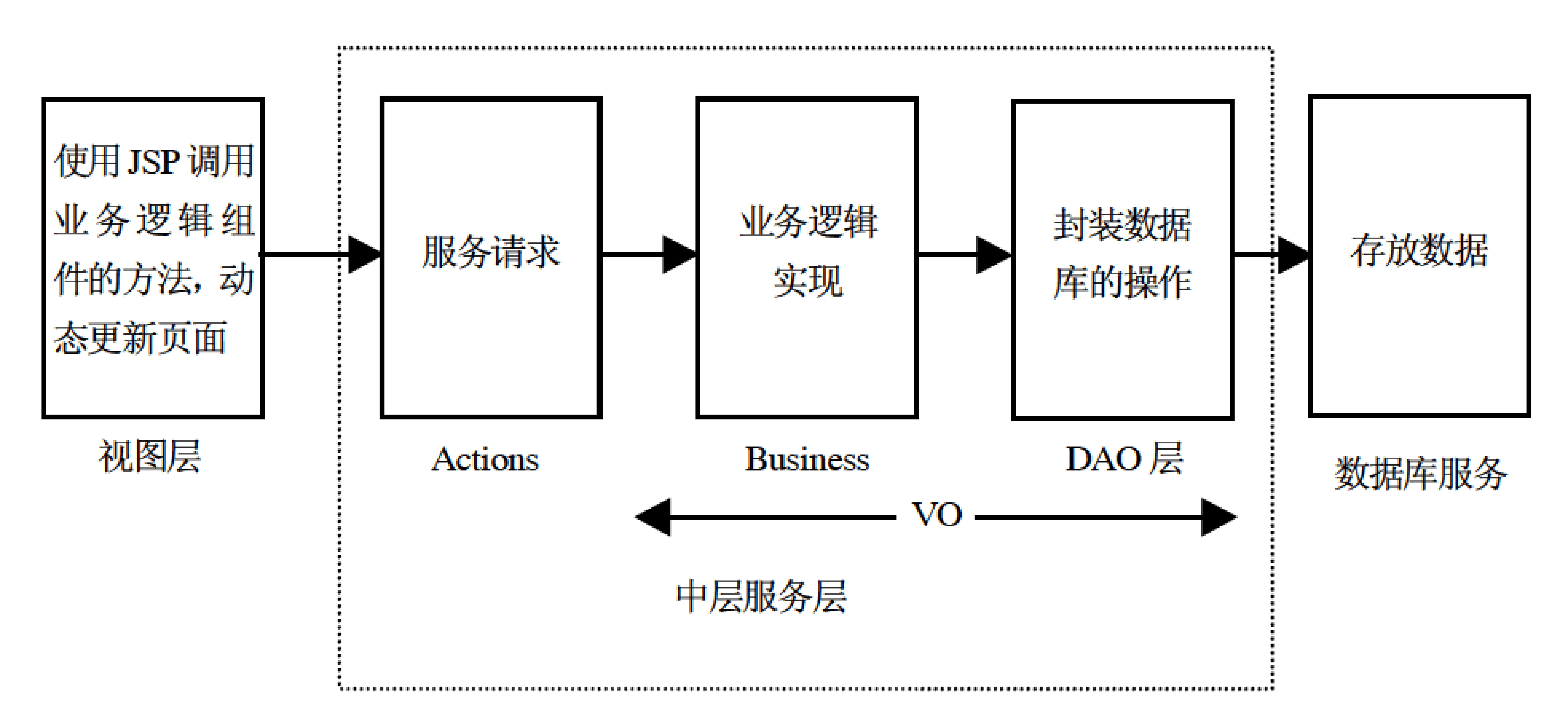


图 11-24　系统的总体设计图

其中包括以下几个主要部分。

- Business（业务逻辑）：负责实现业务逻辑，对 DAO 对象进行封装。
- DAO（数据访问对象）：负责与数据库的交互，封装了数据的增加、删除、修改、查询等操作。
- VO（数值对象）：负责与数据库基表的映射。

11.3　数据库设计

数据库设计主要包括数据库逻辑结构设计、创建数据库、创建表的脚本文件。

11.3.1　数据库逻辑结构设计

数据库设计是系统设计中非常重要的一个环节。数据是一切系统设计的基础，通俗地说，数据库设计就像高楼大厦的根基一样，如果设计不合理、不完善，将在系统开发过程中，甚至到后期的系统维护、功能变更和功能扩充时，引起较多问题，严重时甚至要重新设计，重做大量已完成的工作。

根据功能模块划分的结果可知，本系统的用户有 3 类：管理员、学生和教师。由于管

理员、学生和教师的权限和操作功能大不相同，因此在本系统中需要分别进行数据记录。首先需要如下 3 个数据实体。

- ❑ 管理员数据实体：只需要记录管理员的登录名、姓名和密码，其中登录名和密码是管理功能模块登录验证时所必需的。
- ❑ 学生数据实体：包括学生号、密码、学生姓名、性别、学生所在系、籍贯、联系电话、电子邮件。这些信息中，密码、联系电话和电子邮件由学生自己进行维护，管理员在学生入学时根据填写的信息初始化学生信息，在以后的维护过程中，仅在特殊情况下对信息进行修改操作。
- ❑ 教师数据实体：包括教师号、密码、教师姓名、职称。这些信息由管理员初始化比较好，如果有所改动都要由管理员维护。

除了以上 3 个系统用户实体外，学生课程及成绩管理系统还要对学生课程和班级进行管理，这就需要如下两个数据实体。

- ❑ 课程数据实体：用于记录所有课程的基本信息，包括课程的课程号、课程名、学分、系别和预选修情况。这些信息由学校的工作人员以管理员的身份登录后进行维护。
- ❑ 班级数据实体：用于记录班级的基本信息，包括班级号、教师、课程、教室和上课时间。这些数据由管理员进行录入和维护（如果与学校排课系统等结合，数据就由那些系统来提供）。

以上 5 个实体是基本的数据实体。作为学生课程及成绩管理系统，还要记录学生选课和学分情况，因此又有如下的实体。

学生课程及成绩数据实体：学生号、所上课班级、是否被老师接收和所给学分。

根据以上的分析，设定每一个数据实体都有一个 ID 作为它的唯一标识，那么这 6 个数据实体的关联关系如图 11-25 所示（其中管理员数据实体相对独立，这里没有列出关系图）。

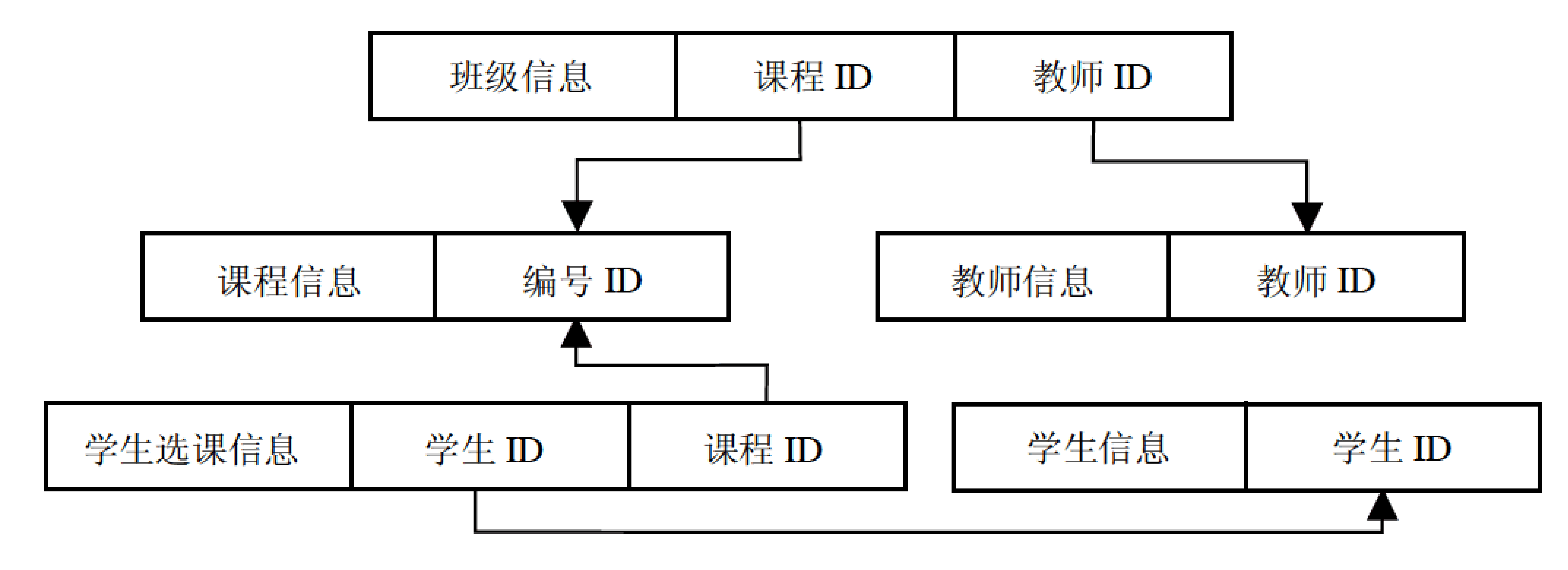


图 11-25　数据实体关系图

基于上面的设计，开始设计表，表与表之间相互关联，共同存储着系统所需要的数据。在设计数据库表的过程中，一般要遵循几条原则：

- ❑ 数据库的一个表最好只存储一个实体或对象的相关信息，不同的实体最好存储在不同的数据表中，如果实体还可以再划分，实体的划分原则是，划分后的实体比当前系统要开发实体的复杂度小。

- ❑ 数据表的信息结构一定要合适，表的字段的数量一般不要过多。
- ❑ 扩充信息和动态变化的信息一定要分别放在不同的表中。
- ❑ 多对多的表关系尽量不出现。

设计数据库表及表间关系，通过图 11-26 表示。

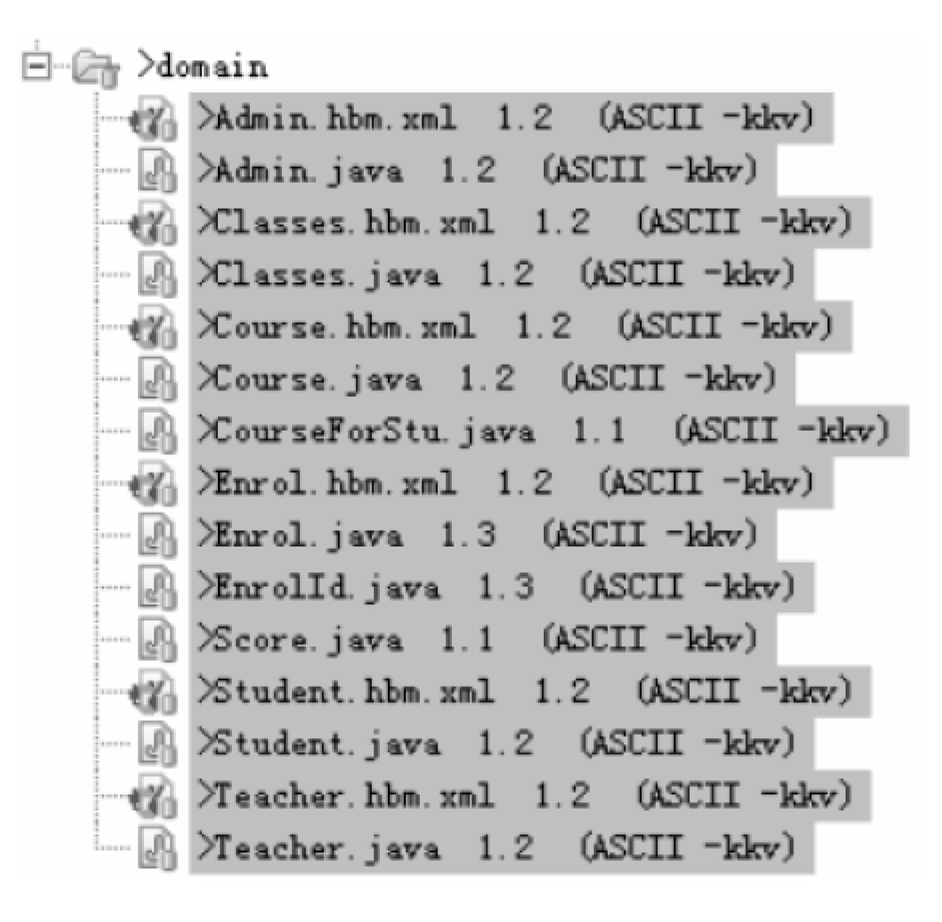


图 11-26　数据库表关系图

11.3.2　创建数据库

首先要创建一个数据库，可以使用 MySQL 的辅助图形化界面工具 SQLyog，这个工具的下载安装已经在前面章节介绍过，如果读者还不熟悉 MySQL 或 SQLyog 这个工具，可以参考前面的相关章节。使用 SQLyog 连接到 MySQL，然后创建数据库 STU，为新建的数据库添加用户并设置权限（选择 Tools→User Manager→Add user 命令或直接按快捷键 Ctrl+U）。可以将对于这个新建数据库的所有权限全都赋给这个用户。接下来要在这个数据库中创建数据表，由前面的分析得知这个系统中需要建立 6 张数据表，分别如下所示。

- ❑ 管理员表（admin）：用于存放管理员用户的数据记录。
- ❑ 学生信息表（student）：用于存放学生的基本信息。
- ❑ 教师信息表（teacher）：用于存放所有上课教师的基本信息。
- ❑ 课程信息表（course）：用于存放所有开课课程的数据记录基本信息。
- ❑ 班级信息表（classes）：用于存放所有与班级相关的信息。
- ❑ 学生课程及成绩表（enrol）：用于存放所有学生课程及成绩信息。

这 6 张数据表的字段说明如表 11-1~表 11-6 所示。

表 11-1　admin 管理员表

序　号	字　段	含　义	类　型
1	id	管理员编号	Int
2	name	姓名	Varchar
3	password	密码	Varchar

表 11-2　student 学生信息表

序　　号	字　　段	含　　义	类　　型
1	id	学生编号	Varchar
2	name	姓名	Varchar
3	password	密码	Varchar
4	jiguan	籍贯	Varchar
5	department	所在系	Varchar
6	sex	性别	Varchar
7	mark	学分	Int
8	tel	联系电话	Varchar
9	email	电子邮箱	Varchar

表 11-3　teacher 教师信息表

序　　号	字　　段	含　　义	类　　型
1	id	教师编号	Varchar
2	name	姓名	Varchar
3	title	职称	Varchar
4	password	密码	Varchar

表 11-4　course 课程信息表

序　　号	字　　段	含　　义	类　　型
1	id	课程编号	Varchar
2	name	课程名	Varchar
3	mark	学分	Int
4	prepare	预选课程	Varchar
5	dep	所在系	Varchar

表 11-5　classes 班级信息表

序　　号	字　　段	含　　义	类　　型
1	id	班级编号	Varchar
2	tea_id	姓名	Varchar
3	cour_id	密码	Varchar
4	room_id	教室号	Varchar
5	cour_time	上课时间	Varchar

表 11-6　enrol 学生课程及成绩信息表

序　　号	字　　段	含　　义	类　　型
1	stu_id	学生编号	Varchar
2	class_id	班级编号	Varchar
3	accept	是否被接收	Varchar
4	score	成绩	Varchar

11.3.3 创建表的脚本文件

根据数据库字段设计，编写的创建数据库表的 SQL 语句如下。

创建数据表 admin 的 SQL 语句：

```
CREATE TABLE 'admin' (
'id' int(16) NOT NULL,
'name' varchar(32) default NULL,
'password' varchar(32) default NULL,
PRIMARY KEY ('id')
)
```

创建数据表 student 的 SQL 语句：

```
CREATE TABLE 'student' (
'id' varchar(32) NOT NULL,
'name' varchar(32) default NULL,
'password' varchar(32) default NULL,
'jiguan' varchar(32) default NULL,
'department' varchar(32) default NULL,
'sex' varchar(32) default NULL,
'mark' int(11) default NULL,
'tel' varchar(32) default NULL,
'email' varchar(32) default NULL,
PRIMARY KEY ('id')
)
```

创建数据表 teacher 的 SQL 语句：

```
CREATE TABLE 'teacher' (
'id' varchar(32) NOT NULL,
'name' varchar(32) default NULL,
'password' varchar(32) default NULL,
'title' varchar(32) default NULL,
PRIMARY KEY ('id')
)
```

创建数据表 course 的 SQL 语句：

```
CREATE TABLE 'course' (
'id' varchar(32) NOT NULL,
'name' varchar(32) default NULL,
'mark' int(11) default NULL,
'prepare' varchar(32) default NULL,
'dep' varchar(32) default NULL,
PRIMARY KEY ('id')
)
```

创建数据表 classes 的 SQL 语句：

```
CREATE TABLE 'classes' (
'id' varchar(32) NOT NULL,
'tea_id' varchar(32) default NULL,
'cour_id' varchar(32) default NULL,
'room_id' varchar(32) default NULL,
'cour_time' varchar(32) default NULL,
PRIMARY KEY ('id'),
KEY 'FK_Reference_4' ('tea_id'),
KEY 'FK_classes' ('cour_id'),
CONSTRAINT 'classes_ibfk_1' FOREIGN KEY ('cour_id') REFERENCES 'course'
('id'),
CONSTRAINT 'FK_Reference_4' FOREIGN KEY ('tea_id') REFERENCES 'teacher'
('id')
)
```

创建数据表 enrol 的 SQL 语句：

```
CREATE TABLE 'enrol' (
'class_id' varchar(32) default NULL,
'stu_id' varchar(32) default NULL,
'accept' varchar(32) default NULL,
'score' varchar(32) default NULL,
KEY 'FK_enrol' ('class_id'),
KEY 'FK_Reference_2' ('stu_id'),
CONSTRAINT 'enrol_ibfk_1' FOREIGN KEY ('class_id') REFERENCES 'classes'
('id'),
CONSTRAINT 'enrol_ibfk_2' FOREIGN KEY ('stu_id') REFERENCES 'student' ('id')
)
```

为了方便后面的开发，在这里先向数据库的 admin 表中插入一条数据，SQL 语句如下：

```
insert into 'stu'.'admin' ('id`, 'name', 'password') values ('1', 'admin',
'admin' )
```

11.4　系统详细设计

11.4.1　界面设计

本系统的界面共分为如下 4 个大的模块。

- 登录模块：此模块是用于不同系统角色的登录，也是系统的唯一入口。
- 管理员模块：此模块是用于管理员对学生、教师、课程和班级等信息的管理和维护，包括如下几个部分。
  - 学生管理：增加、修改、删除学生。
  - 教师管理：增加、修改、删除教师。
  - 课程管理：增加、修改、删除课程。

  - ➢ 班级管理：增加、修改、删除班级。
- ❑ 学生模块：此模块是学生管理操作界面，包括如下几个部分。
  - ➢ 选报课程：查看可选课程、选课。
  - ➢ 查看成绩：查看自己的成绩。
  - ➢ 个人信息：修改。
- ❑ 教师模块：此模块是教师管理操作界面，包括如下几个部分。
  - ➢ 接收学生选课：查看、接收学生。
  - ➢ 打分：查看、打分。

根据这些整体关系设计，下面对每一个部分给出主要的界面及其设计思路。

1．登录界面

系统的任何用户使用系统，都必须从系统的登录界面进入。这是任何一个信息管理系统的保密性的需要。根据之前的需求分析和概要设计可以知道，系统包括 3 类用户：学生、教师和管理员。为了能够让 3 类用户使用同一个页面登录，在登录界面中，提供了选择登录用户类型的下拉列表框，由于使用的是 Struts 的标签，下拉列表框效果并不是很好，但足以满足需要。再加上输入用户名和密码的文本框以及“提交”按钮组成了登录页面。效果如图 11-27 所示。

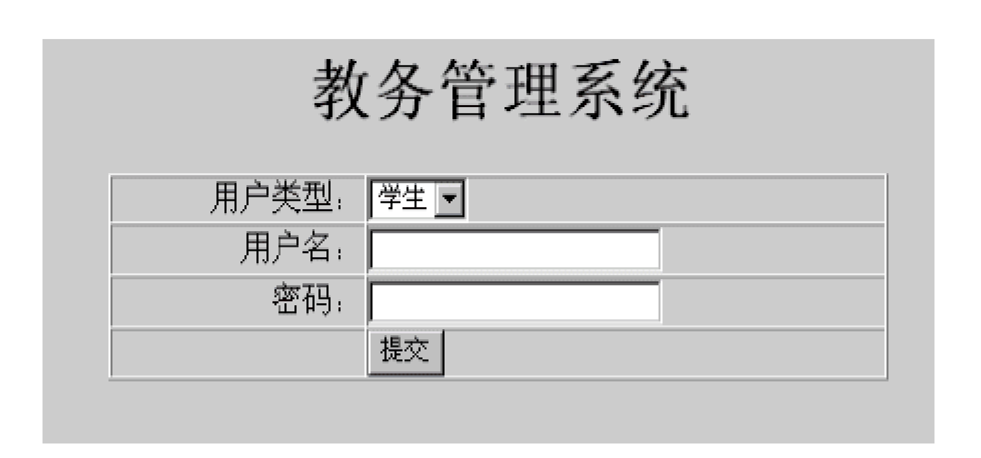


图 11-27　系统登录界面设计

在实现登录界面前，应该先创建一个 Web 服务器默认的欢迎页面 index.jsp，以实现自动跳转到登录页面。index.jsp 的代码如下：

```
<%
response.sendRedirect("login.jsp");
%>
```

接下来创建登录页面，登录页面 login.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
<%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html lang="true">
<head>
<html:base />
<title>login</title>
<style type="text/css">
<!--
```

```
.style1 {
color: #0000FF;
font-weight: bold;
font-size: xx-large;
}
.style2 {color: #FF0000}
-->
</style>
</head><body>
<html:form action="/login" method="post">
<table width="100%" border="0" align="center">
<tr align="center">
<td>
<p class="style1">
学生课程及成绩管理系统
</p>
</tr>
<tr>
<td align="center">
<span class="style2"> <html:errors /> </span>
</tr>
<tr>
<td align="center">
<table border="1">
<tr>
<td align="center">
用户类型:
<td>
<html:select property="sort" multiple="true" size="1" value="1">
<html:option key="a" value="1">学生</html:option>
<html:option key="b" value="2">教师</html:option>
<html:option key="c" value="3">管理员</html:option>
</html:select>
</td>
<tr>
<td width="22%" align="center">
用  户  名:
</td>
<td width="74%">
<html:text property="username" />
<html:errors property="username" />
</td>
</tr>
<tr>
<td align="center">
密        码:
</td>
<td>
<html:password property="password" />
```

```
<html:errors property="password" />
</td>
</tr>
<tr>
<td align="center">  </td>
<td>
<html:submit value="登录" />
</td>
</tr>
</table>
</td>
</tr>
</table>
</html:form>
</body>
</html:html>
```

由于页面中使用到 Struts 标签，所以在代码开始部分导入了 Strutsde 标签库。至此，登录页面已完成。

2．管理员管理首页

管理员登录后，将得到欢迎信息表示登录成功；如果登录失败，则会有错误提示信息。管理员可以通过单击“学生”、“教师”、“课程”和“班级”4 个链接进入不同页面继续下一步的操作，效果如图 11-28 所示。

你已经经过认证登录！单击进入下一步操作！

学生>> 教师>> 课程>> 班级

图 11-28　管理员管理首页

由于这个页面（admin.jsp）的实现简单，这里就不给出代码了。管理首页设计完成后，接下来就要分别开始设计各个功能页面的界面，首先从管理员管理学生的界面开始。

11.4.2　目录和包结构设计

在进行程序设计和开发之前，要设计目录和包的结构。良好的结构会使代码逻辑清楚且容易阅读。一般一个设计良好的结构都有其共同的特点，就是逻辑清楚。本系统的目录结构如图 11-29 所示。

在这个目录结构中，MyStuMan 是项目的根目录，也是项目的名称。其下 src 目录用于存放原文件，所有的 Java 类都定义在这个文件夹下。WebRoot 目录是发布时网站的根目录，其下放置 JSP 页面，WEB-INF 目录下存放系统的配置文件，如 web.xml 等。

这个目录结构是通用的目录结构，读者可以根据需要进行相应的修改。

```
>StuManager [localhost]
  >src
    >com.stuman.dao
    >com.stuman.dao.hibernate
    >com.stuman.dao.imp
    >com.stuman.domain
    >com.stuman.service
    >com.stuman.util
    >com.stuman.web.filters
    >com.stuman.web.struts.action
    >com.stuman.web.struts.form
    >applicationContext0.xml 1.1 (ASCII -kkv)
    >ApplicationResources_zh_CN.properties 1.1 (ASCII
    >ApplicationResources.properties 1.1 (ASCII -kkv)
    >daoconfig.properties 1.1 (ASCII -kkv)
    >ehcache.xml 1.1 (ASCII -kkv)
    >hibernate.cfg.xml 1.1 (ASCII -kkv)
    >log4j.properties 1.1 (ASCII -kkv)
  JRE System Library [JRE1.5]
  J2EE 1.4 Library Container
  Hibernate 3.0 Core Libraries
  >lib
  >WebRoot
    >css
    >images
    >META-INF
    >WEB-INF
    >AddClass.jsp 1.2 (ASCII -kkv)
```

图 11-29 目录及包结构

11.4.3 HibernateUtil 设计

本系统采用 Struts+Hibernate 技术进行开发，由 Hibernate 进行数据对象的操作，这里定义一个 HibernateUtil 类负责初始化 Hibernate。由它创建全局的 SessionFactory 实例，并且提供创建 Session 实例、关闭 Session 实例、打开/关闭事务以及重新创建 SessionFactory 实例的实用方法。而且所有方法均为静态方法。

HibernateUtil 类是用两个 ThreadLocal 类型的属性以保持在一次请求过程中共享单一的 Session 和 Transaction 实例，Session 和 Transaction 实例可以跨越多个一次请求的多个方法，这有助于实现集合属性的延迟加载等 Hibernate 特性。HibernateUtil 代码如下：

```
package com.stuman.dao.hibernate;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
private static final SessionFactory sessionFactory;
static {
try {
//创建 SessionFactory
sessionFactory = new Configuration().configure()
.buildSessionFactory();
} catch (Throwable ex) {
ex.printStackTrace();
System.out.println("Initial SessionFactory creation failed.");
throw new ExceptionInInitializerError(ex);
```

```
}
}
public static final ThreadLocal tLocalsess = new ThreadLocal();
public static final ThreadLocal tLocaltx = new ThreadLocal();
//取得 Session
public static Session currentSession() {
Session session = (Session) tLocalsess.get();
//打开一个新的 Session，如果当前的不可用
try {
if(session == null || !session.isOpen()) session = openSession();
tLocalsess.set(session);
}
} catch (HibernateException e) {
//抛出 HibernateException 异常
e.printStackTrace();
}
return session;
}
//关闭 Session
public static void closeSession() {
Session session = (Session) tLocalsess.get();
tLocalsess.set(null);
try {
if (session != null && session.isOpen()) {
session.close();
}
} catch (HibernateException e) {
//抛出 InfrastructureException 异常
}
}
//开始事务
public static void beginTransaction() {
//声明 Transaction 类型对象 tx，并赋初值
Transaction tx = (Transaction) tLocaltx.get();
try {
if (tx == null) {
tx = currentSession().beginTransaction();
tLocaltx.set(tx);
}
} catch (HibernateException e) {
//抛出 InfrastructureException 异常
}
}
//关闭事务
public static void commitTransaction() {
Transaction tx = (Transaction) tLocaltx.get();
try {
if (tx != null && !tx.wasCommitted() && !tx.wasRolledBack())
tx.commit();
```

```
tLocaltx.set(null);
System.out.println("commit tx");
} catch (HibernateException e) {
//抛出 InfrastructureException 异常
}
}
//事务回滚
public static void rollbackTransaction() {
Transaction tx = (Transaction) tLocaltx.get();
try {
tLocaltx.set(null);
if (tx != null && !tx.wasCommitted() && !tx.wasRolledBack()) {
tx.rollback();
}
} catch (HibernateException e) {
//抛出 InfrastructureException 异常
}
}
private static Session openSession() throws HibernateException {
return getSessionFactory().openSession();
}
private static SessionFactory getSessionFactory() throws HibernateException
{
eturn sessionFactory;/
}
}
```

因为在一次请求中需要共享单一的 Session 和 Transaction 实例，因此不能在每个方法后关闭 Session，应该在一次请求全部处理完成后关闭 Session。为此，可以设计一个过滤器，在过滤器中统一关闭 Session。实现该功能的过滤器代码如下：

```
package com.stuman.dao.hibernate;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.*;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
public class CloseSessionFilter implements Filter {
Log logger = LogFactory.getLog(this.getClass());
protected FilterConfig config;
//初始化方法
public void init(FilterConfig arg0) throws ServletException {
this.config = arg0;
}
//doFilter 方法定义操作
```

```
public void doFilter(
ServletRequest request,
ServletResponse response,
FilterChain chain)
throws IOException, ServletException {
try{
chain.doFilter((HttpServletRequest)request,
(HttpServletResponse)response);
}
finally{
try{
HibernateUtil.commitTransaction();      //提交事务
}catch (Exception e){
HibernateUtil.rollbackTransaction();    //回滚事务
}finally{
HibernateUtil.closeSession();           //关闭 Session
}
}
}
//destroy 方法
public void destroy() {
this.config = null;
}
}
```

这样一来，在使用 Hibernate 时更加方便，不用每次为创建 SessionFactory 实例、Session 实例或是关闭 Session 实例以及打开/关闭事务等操作单独写代码，只要调用上面类的方法即可，从而简化了操作。

11.4.4　SetCharacterEncodingFilter 设计

在进行 Web 开发时，经常会遇到中文显示出现乱码的情况，这是因为 Java 内置的字符集与页面显示的字符集不一致造成的。为了解决这个问题，开发人员需要转化字符编码，但是如果对所有的输入输出信息都做编码转化显然比较麻烦。通过定义一个 Filter 来自动地实现字符编码的转化是一个比较好的方法。而且实现并不复杂，实现步骤如下：

（1）定义 SetCharacterEncodingFilter 类

将这个类放在 com.stuman.util 包下，这个类要实现 Filter 接口，代码如下：

```
package com.stuman.util;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class SetCharacterEncodingFilter implements Filter {
```

```
protected String encoding = null;
protected FilterConfig filterConfig = null;
protected boolean ignore = true;
//destroy 方法
public void destroy() {
this.encoding = null;
this.filterConfig = null;
}
//选择设置使用的字符编码
public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
throws IOException, ServletException {
//选择使用的字符编码
if (ignore || (request.getCharacterEncoding() == null)) {
String encoding = selectEncoding(request);
if (encoding != null)
request.setCharacterEncoding(encoding);
}
//将控制权交给下一个 Filter
chain.doFilter(request, response);
}
//将这个 Filter 放置在服务中
public void init(FilterConfig filterConfig) throws ServletException {
this.filterConfig = filterConfig;
this.encoding = filterConfig.getInitParameter("encoding");
String value = filterConfig.getInitParameter("ignore");
if (value == null)
this.ignore = true;
else if (value.equalsIgnoreCase("true"))
this.ignore = true;
else if (value.equalsIgnoreCase("yes"))
this.ignore = true;
else
this.ignore = false;
}
//选择适当的字符编码
protected String selectEncoding(ServletRequest request) {
return (this.encoding);
}
}
```

（2）配置这个 Filter

在 web.xml 文件中添加如下代码：

```
<filter>
<filter-name>SetCharsetEncodingFilter</filter-name>
<filter-class>com.stuman.util.SetCharacterEncodingFilter</filter-class>
<init-param>
<param-name>encoding</param-name>
<param-value>utf-8</param-value>
```

```
</init-param>
</filter>
<filter-mapping>
<filter-name>SetCharsetEncodingFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

通过配置 Filter，即可实现编码的自动转换。

11.4.5　数据层设计

一般将业务逻辑分为服务层逻辑和持久化层逻辑。在实现业务层逻辑时，需要尽可能地保持层次之间的松散耦合，面向接口编程是业务逻辑设计一个最重要的原则。

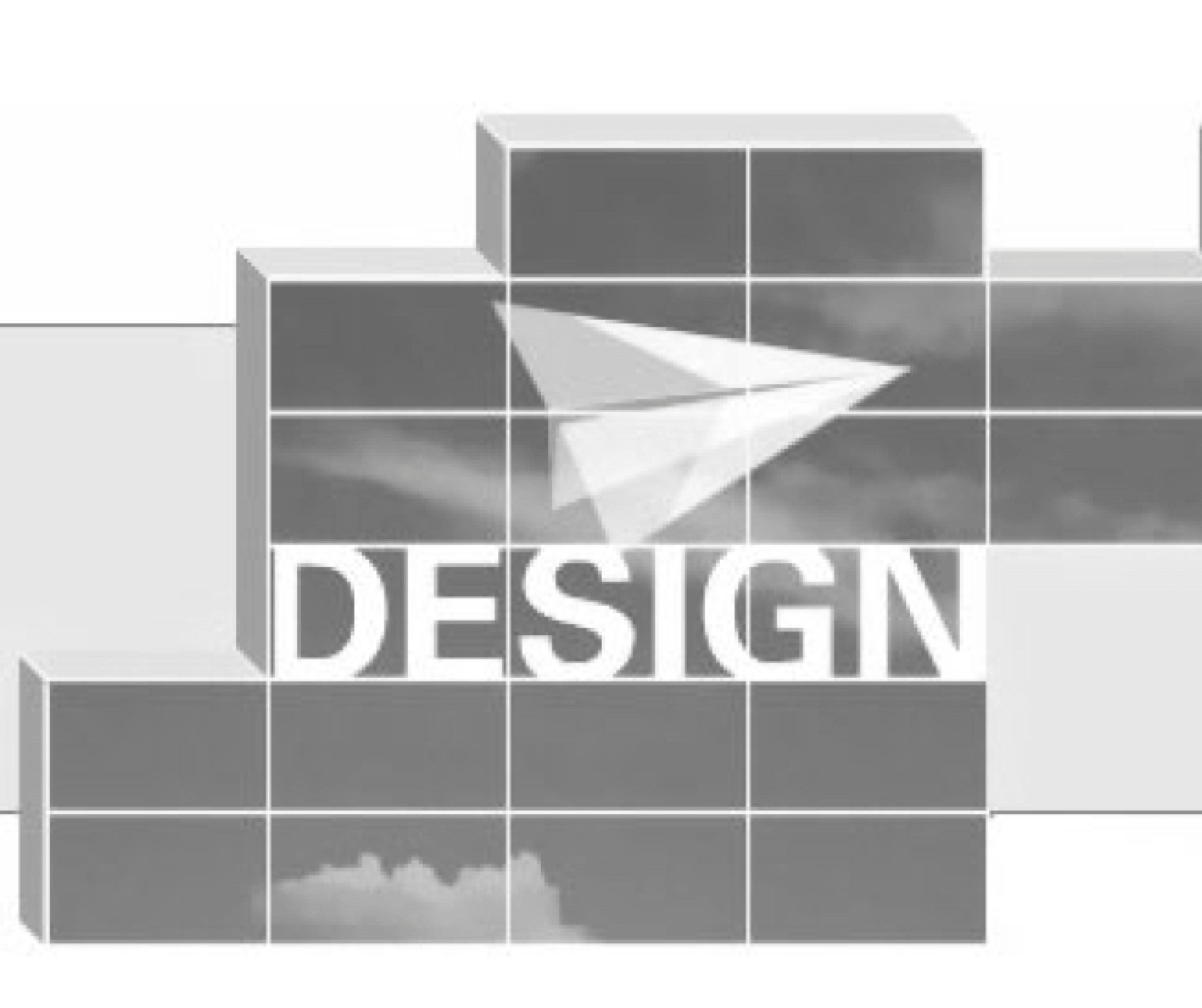
本系统的持久化逻辑采用 Hibernate 作为中间件，并使用 DAO 设计模式实现。DAO 模式是 J2EE 核心模式中的一种，主要是在业务核心方法和具体数据源之间增加一层，这样就减少了两者的耦合。因为具体持久层数据源可能是多样化的，例如可能是 XML 或者是关系数据库，在具体的关系数据库中，也可能有不同的产品如 Oracle 或 MySQL（当然在本系统的开发过程中使用 MySQL），通过使用 DAO 模式，业务核心部分就无须涉及如何具体操作数据库，每个持久化类对应一个 DAO，它实现了持久化类的创建、查询、更新及删除方法，即 CRUD（CREATE、RETRIEVE、UPDATE、DELETE）方法，以及其他访问持久化机制的方法。在相应的 DAO 实现中，调用 Hibernate API 访问持久层。这样只有特定于 Hibernate 的 DAO 实现需要依赖 Hibernate API，当改用其他的持久化机制或持久化中间件时，只需要创建新的 DAO 实现，无须更改应用中其他业务逻辑代码。这便是 DAO 模式的优点。

11.5　小　　结

本章详细地讲述了利用 J2EE 技术构建一个比较通用的教务管理系统。通过本章的学习，读者对 Java B/S 结构的软件开发有了一定的认识。读者可以在本系统源代码基础上进行修改，以符合实际项目的需要。

将源代码中提供的本章应用程序(整个目录 StudentManager)复制到本机所安装的\Apache Software Foundation\Tomcat 5.5\webapps 路径下即可运行本章的案例。

第 4 篇



高级开发技术

本篇将介绍 MySQL 的一些高级特性及 XML 技术，主要内容包括 MySQL 5.0 存储过程的新特性、新 SQL 语句和 Loops 循环语句、数据导入导出工具 mysqlimport、MySQL 性能的优化以及 XML 与 MySQL 的结合使用等，掌握这些高级开发技术将有助于更好地运用 MySQL 开发出功能强大、性能稳定的应用系统。

第 12 章　MySQL 5.0 高级特性

本章主要介绍 MySQL 5.0 的高级特性，包括新 SQL 语句和 Loops 循环语句，数据导入导出工具 mysqlimport 的语法和常用选项介绍，最后还介绍了 MySQL 进行性能优化关键系统参数。

12.1　MySQL 5.0 存储过程新特性

MySQL 5.0 新特性是为需要了解 5.0 版本新特性的 MySQL 老用户而言的，介绍包括调用存储过程、特征子句和参数等内容。

12.1.1　存储过程体中合法的 MySQL 语句

什么样的 SQL 语句在 MySQL 存储过程中才是合法的呢？可以创建一个包含 INSERT、UPDATE、DELETE、SELECT、DROP、CREATE 和 REPLACE 等的语句。唯一需要记住的是如果代码中包含 MySQL 扩充功能，那么代码将不能移植。在标准 SQL 语句中：任何数据库定义语言都是合法的。如：

```
CREATE PROCEDURE p () DELETE FROM t;
```

SET、COMMIT 以及 ROLLBACK 也是合法的。如：

```
CREATE PROCEDURE p () SET @x = 5;
```

MySQL 的附加功能：任何数据操作语言的语句都将合法。如：

```
CREATE PROCEDURE p () DROP TABLE t;
```

MySQL 的扩充功能：直接的 SELECT 也是合法的。如：

```
CREATE PROCEDURE p () SELECT 'a';
```

顺便提一下，这里将存储过程中包括 DDL 语句的功能称为 MySQL 附加功能的原因是在 SQL 标准中把这个定义为非核心的，即可选组件。在过程体中有一个约束，就是不能有对例程或表操作的数据库操作语句。例如下面的例子就是非法的：

```
CREATE PROCEDURE p1 ()
CREATE PROCEDURE p2 () DELETE FROM t;
```

下面这些对 MySQL 5.0 来说是全新的语句，过程体中是非法的：CREATE PROCEDURE、

ALTER PROCEDURE、DROP PROCEDURE、CREATE FUNCTION、DROP FUNCTION、CREATE TRIGGER、DROP TRIGGER。

不过可以使用"CREATE PROCEDURE db5.p1 () DROP DATABASE db5"，但类似"USE database"语句也是非法的，因为 MySQL 假定默认数据库就是过程的工作场所。

12.1.2　Call the Procedure 调用存储过程

现在即可调用一个存储过程了，需要输入的全部就是 CALL 和过程名以及一个括号再一次强调，括号是必需的，当调用例子中的 p1 过程时，结果是屏幕返回了 t 表的内容。

```
mysql> CALL p1()
+------+
| s1 |
+------+
| 5 |
+------+
1 row in set (0.03 sec)
Query OK, 0 rows affected (0.03 sec)
```

因为过程中的语句是"SELECT * FROM t;"，其他实现方式 mysql> CALL p1()和下面语句的执行效果一样：

```
mysql> SELECT * FROM t;
```

所以，调用 p1 过程就相当于执行了语句："SELECT * FROM t;"。

12.1.3　Characteristics Clauses 特征子句

CREATE PROCEDURE p2 ()

```
LANGUAGE SQL <--
NOT DETERMINISTIC <--
SQL SECURITY DEFINER <--
COMMENT 'A Procedure' <--
SELECT CURRENT_DATE, RAND() FROM t
```

这里给出的是一些能反映存储过程特性的子句。子句内容在括号之后、主体之前，这些子句都是可选的。

```
LANGUAGE SQL <--
NOT DETERMINISTIC
SQL SECURITY DEFINER
COMMENT 'A Procedure'
SELECT CURRENT_DATE, RAND() FROM t
```

这个 LANGUAGE SQL 子句是没有作用的。仅是为了说明下面过程的主体使用 SQL 语言编写。这条是系统默认的，但在这里声明是有用的，因为某些 DBMS（IBM 的 DB2）需

要它，如果关注 DB2 的兼容问题最好还是用上。此外，今后可能会出现除 SQL 外的其他语言支持的存储过程。

```
LANGUAGE SQL
NOT DETERMINISTIC <--
SQL SECURITY DEFINER
COMMENT 'A Procedure'
SELECT CURRENT_DATE, RAND() FROM t
```

下一个子句 NOT DETERMINISTIC，是传递给系统的信息。这里一个确定过程的定义就是那些每次输入一样输出也一样的程序。在该例中，既然主体中含有 SELECT 语句，那返回肯定是未知的，因此称其为 NOT DETERMINISTIC。但是 MySQL 内置的优化程序不会注意这个，至少在现在不注意。

```
LANGUAGE SQL
NOT DETERMINISTIC
SQL SECURITY DEFINER <--
COMMENT 'A Procedure'
SELECT CURRENT_DATE, RAND() FROM t
```

下一个子句是 SQL SECURITY，可以定义为 SQL SECURITY DEFINER 或 SQL SECURITY INVOKER。

这就进入了权限控制的领域，当然在后面将会有测试权限的例子。

SQL SECURITY DEFINER 意味着在调用时检查创建过程用户的权限（另一个选项是 SQL SECURITY INVOKER）。

此时，使用 SQL SECURITY DEFINER 指令告诉 MySQL 服务器检查创建过程的用户即可，当过程已经被调用，就不检查执行调用过程的用户了。而另一个选项（INVOKER）则是告诉服务器在这一步仍然要检查调用者的权限。

```
LANGUAGE SQL
NOT DETERMINISTIC
SQL SECURITY DEFINER
COMMENT 'A Procedure' <--
SELECT CURRENT_DATE, RAND() FROM t
```

COMMENT 'A procedure'是一个可选的注释说明。最后，注释子句会和过程定义存储在一起。这个没有固定的标准。

```
LANGUAGE SQL
NOT DETERMINISTIC
SQL SECURITY DEFINER
COMMENT ''
SELECT CURRENT_DATE, RAND() FROM t
```

上面过程和下面语句是等效的：

```
CREATE PROCEDURE p2 ()
SELECT CURRENT_DATE, RAND() FROM t
```

特征子句也有默认值，如果省略了就相当于：

```
LANGUAGE SQL NOT DETERMINISTIC SQL SECURITY DEFINER COMMENT ''
```

Digression：（调用 p2()的结果）

```
mysql> call p2()
+--------------+-----------------+
| CURRENT_DATE | RAND() |
+--------------+-----------------+
| 2004-11-09 | 0.7822275075896 |
+--------------+-----------------+
1 row in set (0.26 sec)
Query OK, 0 rows affected (0.26 sec)
```

当调用过程 p2 时，一个 SELECT 语句被执行返回期望获得的随机数。

Digression：sql_mode unchanging（不会改变的）

```
sql_mode
mysql> set sql_mode='ansi'
mysql> create procedure p3()select'a'||'b'
mysql> set sql_mode=''
mysql> call p3()
+------------+
| 'a' || 'b' |
+------------+
| ab |
+------------+
```

MySQL 在过程创建时会自动保持运行环境。例如，需要使用两条竖线来连接字符串，但是，这只有在 sql mode 为 ansi 时才合法。如果将 sql mode 改为 non-ansi，不用担心，它仍然能工作，只要它第一次使用时能正常工作。

12.1.4　Parameters 参数

下面更进一步地研究怎样在存储过程中定义参数。

（1）CREATE PROCEDURE p5

() ...

（2）CREATE PROCEDURE p5

([IN] name data-type) ...

（3）CREATE PROCEDURE p5

(OUT name data-type) ...

（4）CREATE PROCEDURE p5

(INOUT name data-type) ...

回忆前面讲过的参数列表必须在存储过程名后的括号中。上面的第一个例子中的参数列表是空的，第二个例子中有一个输入参数。这里的词 IN 可选，因为默认参数为

IN（input）。

第三个例子中有一个输出参数，第四个例子中有一个参数，既能作为输入也可以作为输出。

- ❑ IN example（输入的例子）

```
mysql> CREATE PROCEDURE p5(p INT) SET @x = p
Query OK, 0 rows affected (0.00 sec)
mysql> CALL p5(12345)
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @x
+--------+
| @x |
+--------+
| 12345 |
+--------+
1 row in set (0.00 sec)
```

这个 IN 的例子演示的是有输入参数的过程。在过程体中将会话变量 x 设定为参数 p 的值。然后调用过程，将 12345 传入参数 p。选择显示会话变量@x，证明已经将参数值 12345 传入。

- ❑ OUT example（输出的例子）

```
mysql> CREATE PROCEDURE p6 (OUT p INT)
-> SET p = -5
mysql> CALL p6(@y)
mysql> SELECT @y
+------+
| @y |
+------+
| -5 |
+------+
```

这是另一个例子。这次的 p 是输出参数，然后在过程调用中将 p 的值传入会话变量@y 中。

在过程体中，给参数赋值-5，在调用后可以看出，OUT 是告诉 DBMS 值是从过程中传出的。同样可以用语句"SET @y = -5;"来达到和 Compound Statements 复合语句同样的效果。

现在详细分析过程体：

```
CREATE PROCEDURE p7 ()
BEGIN
SET @a = 5;
SET @b = 5;
INSERT INTO t VALUES (@a);
SELECT s1 * @a FROM t WHERE s1 >= @b;
END; /* I won't CALL this.这个语句将不会被调用*/
```

完成过程体的构造就是 BEGIN/END 块。这个 BEGIN/END 语句块和 Pascal 语言中的

BEGIN/END 是基本相同的，和 C 语言的框架是很相似的。可以使用块去封装多条语句。在这个例子中，使用了多条设定会话变量的语句，然后完成了一些 INSERT 和 SELECT 语句。如果过程体中有多条语句，那么就需要 BEGIN/END 块了。BEGIN/END 块也被称为复合语句，在这里可以进行变量定义和流程控制。

12.2　新 SQL 语句和 Loops 循环语句

介绍包括新 SQL 语句、Conditions and IF-THEN-ELSE 条件式、IF-THEN-ELSE 和循环语句等内容。

12.2.1　新 SQL 语句

在复合语句中声明变量的指令是 DECLARE。

（1）两个 DECLARE 语句的例子

```
CREATE PROCEDURE p8 ()
BEGIN
DECLARE a INT;
DECLARE b INT;
SET a = 5;
SET b = 5;
INSERT INTO t VALUES (a);
SELECT s1 * a FROM t WHERE s1 >= b;
END; /* I won't CALL this */
```

在过程中定义的变量并不是真正的定义，只是在 BEGIN/END 块中定义了而已（译注：也就是形参）。注意这些变量和会话变量不一样，不能使用修饰符@，必须清楚地在 BEGIN/END 块中声明变量和它们的类型。变量一旦声明，就能在任何能使用会话变量、文字、列名的地方使用。

（2）没有默认子句和设定语句的例子

```
CREATE PROCEDURE p9 ()
BEGIN
DECLARE a INT /* there is no DEFAULT clause */;
DECLARE b INT /* there is no DEFAULT clause */;
SET a = 5; /* there is a SET statement */
SET b = 5; /* there is a SET statement */
INSERT INTO t VALUES (a);
SELECT s1 * a FROM t WHERE s1 >= b;
END; /* I won't CALL this */
```

有很多初始化变量的方法。如果没有默认的子句，那么变量的初始值为 NULL。可以在任何时候使用 SET 语句给变量赋值。

（3）含有 DEFAULT 子句的例子

```
CREATE PROCEDURE p10 ()
BEGIN
DECLARE a, b INT DEFAULT 5;
INSERT INTO t VALUES (a);
SELECT s1 * a FROM t WHERE s1 >= b;
END;
```

这里做了一些改变，但是结果还是一样的。在这里使用了 DEFAULT 子句来设定初始值，这就不需要把 DECLARE 和 SET 语句的实现分开了。

（4）调用的例子

```
mysql> CALL p10()
+--------+
| s1 * a |
+--------+
| 25 |
| 25 |
+--------+
2 rows in set (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
```

结果显示了过程能正常工作。

（5）作用域

```
CREATE PROCEDURE p11 ()
BEGIN
DECLARE x1 CHAR(5) DEFAULT 'outer';
BEGIN
DECLARE x1 CHAR(5) DEFAULT 'inner';
SELECT x1;
END;
SELECT x1;
END;
```

现在讨论作用域的问题。例子中有嵌套的 BEGIN/END 块，当然这是合法的。同时包含两个变量，名字都是 x1，这样也是合法的。内部的变量在其作用域内享有更高的优先权。当执行到 END 语句时，内部变量消失，此时已经在其作用域外，变量不再可见了，因此在存储过程外再也不能找到这个声明了的变量，但是可以通过 OUT 参数或者将其值指派给会话变量来保存其值。

调用作用域例子的过程：

```
mysql> CALL p11()
+-------+
| x1 |
+-------+
| inner |
+-------+
```

```
+--------+
| x1  |
+--------+
| outer |
+--------+
```

看到的结果是第一个 SELECT 语句检索到最内层的变量，第二个检索到第二层的变量。

12.2.2　Conditions and IF-THEN-ELSE　条件式和 IF-THEN-ELSE

包含条件式的例子

（1）CREATE PROCEDURE p12 (IN parameter1 INT)

```
BEGIN
DECLARE variable1 INT;
SET variable1 = parameter1 + 1;
IF variable1 = 0 THEN
INSERT INTO t VALUES (17);
END IF;
IF parameter1 = 0 THEN
UPDATE t SET s1 = s1 + 1;
ELSE
UPDATE t SET s1 = s1 + 2;
END IF;
END;
```

这里是一个包含 IF 语句的过程。里面有两个 IF 语句，一个是 IF 语句 END IF，另一个是 IF 语句 ELSE 语句 END IF。

```
BEGIN
DECLARE variable1 INT;
SET variable1 = parameter1 + 1; <--
IF variable1 = 0 THEN
INSERT INTO t VALUES (17);
END IF;
IF parameter1 = 0 THEN
UPDATE t SET s1 = s1 + 1;
ELSE
UPDATE t SET s1 = s1 + 2;
END IF;
END;
```

这里变量 variable1 被赋值为 parameter1 加 1 的值，所以执行后变量 variable1 为 1。

```
BEGIN
DECLARE variable1 INT;
SET variable1 = parameter1 + 1;
IF variable1 = 0 THEN <--
INSERT INTO t VALUES (17);
```

```
END IF;
IF parameter1 = 0 THEN
UPDATE t SET s1 = s1 + 1;
ELSE
UPDATE t SET s1 = s1 + 2;
END IF;
END;
因为变量 variable1 值为 1，因此条件"if variable1 = 0"为假，
IF
...
END IF 被跳过，没有被执行。
```

```
BEGIN
DECLARE variable1 INT;
SET variable1 = parameter1 + 1;
IF variable1 = 0 THEN
INSERT INTO t VALUES (17);
END IF;
IF parameter1 = 0 THEN <--
UPDATE t SET s1 = s1 + 1;
ELSE
UPDATE t SET s1 = s1 + 2;
END IF;
END;
```

到第二个 IF 条件，判断结果为真，于是中间语句被执行。

```
BEGIN
DECLARE variable1 INT;
SET variable1 = parameter1 + 1;
IF variable1 = 0 THEN
INSERT INTO t VALUES (17);
END IF;
IF parameter1 = 0 THEN
UPDATE t SET s1 = s1 + 1; <--
ELSE
UPDATE t SET s1 = s1 + 2;
END IF;
END;
```

因为参数 parameter1 值等于 0，UPDATE 语句被执行。如果 parameter1 值为 NULL，则下一条 UPDATE 语句将被执行，现在表 t 中有两行，它们都包含值 5，所以如果调用 p12，两行的值会变成 6。

（2）CALL p12 (0)

调用该过程，传入值为 0，这样 parameter1 的值将为 0。

（3）mysql> CALL p12(0)

```
Query OK, 2 rows affected (0.28 sec)
mysql> SELECT * FROM t
+------+
```

```
| s1 |
+------+
| 6 |
| 6 |
+------+
2 rows in set (0.01 sec)
```

结果也是所期望的那样。

12.2.3　循环语句

有 3 种标准的循环方式：WHILE 循环、LOOP 循环以及 REPEAT 循环。还有一种非标准的循环方式：GO TO，由于 GO TO 容易使流程混乱，建议最好不要用。

（1）WHILE…END WHILE

```
CREATE PROCEDURE p14 ()
BEGIN
DECLARE v INT;
SET v = 0;
WHILE v < 5 DO
INSERT INTO t VALUES (v);
SET v = v + 1;
END WHILE;
END;
```

这是 WHILE 循环的方式，它和 IF 语句相似，因此不需要掌握很多新的语法。这里的 INSERT 和 SET 语句在 WHILE 和 END WHILE 之间，当变量 v 大于 5 时循环将会退出。使用"SET v = 0;"语句是为了防止一个常见的错误，如果没有初始化，默认变量值为 NULL，而 NULL 和任何值操作结果都为 NULL。

❑　WHILE…END WHILE 例子

```
mysql> CALL p14()
Query OK, 1 row affected (0.00 sec)
```

以上就是调用过程 p14 的结果，不用关注系统返回是"one row affected"还是"five rows affected"，因为这里的计数只对最后一个 INSERT 动作进行计数。

❑　WHILE…END WHILE example: CALL

```
mysql> select * from t;
+------+
| s1 |
+------+
......
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
```

```
+-------+
9 rows in set (0.00 sec)
```

调用后可以看到程序向数据库中插入了 5 行。

（2）REPEAT…END REPEAT

```
CREATE PROCEDURE p15 ()
BEGIN
DECLARE v INT;
SET v = 0;
REPEAT
INSERT INTO t VALUES (v);
SET v = v + 1;
UNTIL v >= 5
END REPEAT;
END;
```

这是一个 REPEAT 循环的例子，功能和前面 WHILE 循环一样。区别在于它在执行后检查结果，而 WHILE 则是执行前检查。

- REPEAT…END REPEAT: UNTIL 的作用

```
CREATE PROCEDURE p15 ()
BEGIN
DECLARE v INT;
SET v = 0;
REPEAT
INSERT INTO t VALUES (v);
SET v = v + 1;
UNTIL v >= 5 <--
END REPEAT;
END;
```

注意到 UNTIL 语句后面没有分号，在这里可以不写分号，当然加上额外的分号更好。

- REPEAT…END REPEAT

```
mysql> CALL p15()
Query OK, 1 row affected (0.00 sec)
mysql> SELECT COUNT(*) FROM t
+----------+
| COUNT(*) |
+----------+
| 14 |
+----------+
1 row in set (0.00 sec)
```

可以看到调用 p15 过程后又插入了 5 行记录。

（3）LOOP…END LOOP

```
CREATE PROCEDURE p16 ()
BEGIN
```

```
DECLARE v INT;
SET v = 0;
loop_label: LOOP
INSERT INTO t VALUES (v);
SET v = v + 1;
IF v >= 5 THEN
LEAVE loop_label;
END IF;
ED LOOP;
END;
```

以上是 LOOP 循环的例子。LOOP 循环不需要初始条件，这点和 WHILE 循环相似，同时它又和 REPEAT 循环一样也不需要结束条件。

（4）LOOP…END LOOP: with IF and LEAVE（包含 IF 和 LEAVE 的 LOOP 循环）

```
CREATE PROCEDURE p16 ()
BEGIN
DECLARE v INT;
SET v = 0;
loop_label: LOOP
INSERT INTO t VALUES (v);
SET v = v + 1;
IF v >= 5 THEN <--
LEAVE loop_label;
END IF;
END LOOP;
END;
```

在循环内部加入 IF 语句，在 IF 语句中包含 LEAVE 语句。这里 LEAVE 语句的意义是离开循环。

LEAVE 的语法是 LEAVE 加循环语句标号，关于循环语句的标号问题会在后面进一步讲解。

（5）LOOP…END LOOP: calling

```
mysql> CALL p16()
Query OK, 1 row affected (0.00 sec)
mysql> SELECT COUNT(*) FROM t
+----------+
| COUNT(*) |
+----------+
| 19 |
+----------+
1 row in set (0.00 sec)
```

调用过程 p16 后，结果是另 5 行被插入表 t 中。

（6）Labels（标号）

```
CREATE PROCEDURE p17 ()
label_1: BEGIN
```

```
label_2: WHILE 0 = 1 DO LEAVE label_2; END
WHILE;
label_3: REPEAT LEAVE label_3; UNTIL 0 =0
END REPEAT;
label_4: LOOP LEAVE label_4; END LOOP;
END;
```

最后一个循环例子中使用了语句标号。现在这里有一个包含 4 个语句标号的过程的例子。可以在 BEGIN、WHILE、REPEAT 或者 LOOP 语句前使用语句标号，语句标号只能在合法的语句前面使用。因此"LEAVE label_3"意味着离开语句标号名定义为 label_3 的语句或复合语句。

（7）End Labels（标号结束符）

```
CREATE PROCEDURE p18 ()
label_1: BEGIN
label_2: WHILE 0 = 1 DO LEAVE label_2; END
WHILE label_2;
label_3: REPEAT LEAVE label_3; UNTIL 0 =0
END REPEAT label_3 ;
label_4: LOOP LEAVE label_4; END LOOP
label_4 ;
END label_1 ;
```

也可以在语句结束时使用语句标号，和在开头时使用一样。这些标号结束符并不是十分有用。它们是可选的。如果需要，它们必须和开始定义的标号名字一样，当然，为了有良好的编程习惯，方便他人阅读，最好还是使用标号结束符。

（8）LEAVE and Labels（跳出和标号）

```
CREATE PROCEDURE p19 (parameter1 CHAR)
label_1: BEGIN
label_2: BEGIN
label_3: BEGIN
IF parameter1 IS NOT NULL THEN
IF parameter1 = 'a' THEN
LEAVE label_1;
ELSE BEGIN
IF parameter1 = 'b' THEN
LEAVE label_2;
ELSE
LEAVE label_3;
END IF;
END;
END IF;
END IF;
END;
END;
END;
LEAVE
```

语句使程序跳出复杂的复合语句。

ITERATE 迭代：如果目标是 ITERATE（迭代）语句，就必须用到 LEAVE 语句，代码如下：

```
CREATE PROCEDURE p20 ()
BEGIN
DECLARE v INT;
SET v = 0;
loop_label: LOOP
IF v = 3 THEN
SET v = v + 1;
ITERATE loop_label;
END IF;
INSERT INTO t VALUES (v);
SET v = v + 1;
IF v >= 5 THEN
LEAVE loop_label;
END IF;
END LOOP;
END;
```

12.3　数据导入导出工具 mysqlimport

（1）mysqlimport 的语法介绍

mysqlimport 位于 mysql/bin 目录中，是 mysql 的一个载入（或者说导入）数据的一个非常有效的工具。这是一个命令行工具。有两个参数以及大量的选项可供选择。这个工具把一个文本文件（text file）导入到指定的数据库和表中。例如，要从文件 Customers.txt 中把数据导入到数据库 Meet_A_Geek 中的表 Custermers 中：

```
mysqlimport Meet_A_Geek Customers.txt
```

注意，这里 Customers.txt 是要导入数据的文本文件，而 Meet_A_Geek 是要操作的数据库，数据库中的表名是 Customers，这里文本文件的数据格式必须与 Customers 表中的记录格式一致，否则 mysqlimport 命令将会出错。其中表的名字是导入文件的第一个句号（.）前面的文件字符串。

另外一个例子：

```
mysqlimport Meet_A_Geek Cus.to.mers.txt
```

将把文件中的内容导入到数据库 Meet_A_Geek 中的 Cus 表中。

上面的例子中，都只用到两个参数，并没有用到更多的选项，下面介绍 mysqlimport 的选项。

（2）mysqlimport 的常用选项介绍

- -d or --delete：新数据导入数据表中之前删除数据表中的所有信息。

- -f or --force：不管是否遇到错误，mysqlimport 将强制继续插入数据。
- -i or --ignore mysqlimport：跳过或者忽略那些有相同唯一关键字的行，导入文件中的数据将被忽略。
- -l or -lock-tables：数据被插入之前锁住表，这样就防止了在更新数据库时，用户的查询和更新受到影响。
- -r or -replace：这个选项与--i 选项的作用相反；此选项将替代表中有相同唯一关键字的记录。
- --fields-enclosed- by= char：指定文本文件中数据的记录是以什么括起的，很多情况下数据以双引号括起。默认的情况下数据是没有被字符括起的。
- --fields-terminated- by=char：指定各个数据的值之间的分隔符，在句号分隔的文件中，分隔符是句号。可以用此选项指定数据之间的分隔符。默认的分隔符是跳格符（Tab）。
- --lines-terminated- by=str：指定文本文件中行与行之间数据的分隔字符串或者字符。默认的情况下 mysqlimport 以 newline 为行分隔符。可以选择用一个字符串来替代一个单个的字符：一个新行或者一个回车。

mysqlimport 命令常用的选项还有-v 显示版本（version），-p 提示输入密码（password）等。

例如，导入一个以逗号为分隔符的文件，文件中行的记录格式如下：

"1", "ORD89876", "1 Dozen Roses", "19991226"

我们的任务是要把这个文件中的数据导入到数据库 Meet_A_Geek 中的表格 Orders 中，使用这个命令：

bin/mysqlimport –prl –fields-enclosed-by=" –fields-terminated-by=, Meet_A_Geek Orders.txt

该命令可能看起来比较复杂，不过当熟悉了之后，这是非常简单的。第一部分 bin/mysqlimport，告诉操作系统要运行的命令是 mysql/bin 目录下的 mysqlimport，选项 p 是要求输入密码，这样就要求在改动数据库之前输入密码，操作起来会更安全。用了 r 选项是因为想要把表中的唯一关键字与文件记录中有重复唯一关键字的记录替换成文件中的数据。表单中的数据不是最新的，需要用文件中的数据去更新，因而就用 r 这个选项，替代数据库中已经有的记录。l 选项的作用是在插入数据时锁住表，这样就阻止了用户在更新表时对表进行查询或者更改的操作。

12.4　MySQL 性能优化

对于访问量很大的应用，例如日 20 万人次以上的 Web 站点，MySQL 默认的系统参数很难保证 MySQL 运行得非常顺畅。对 MySQL 进行性能优化以下系统参数是比较关键的。

（1）back_log

要求 MySQL 能有的连接数量。当主要 MySQL 线程在一个很短时间内得到非常多的连接请求时起作用，然后主线程花些时间（尽管很短）检查连接并且启动一个新线程。

back_log 值指出在 MySQL 暂时停止回答新请求之前的短时间内多少个请求可以被存在堆栈中。如果期望在一个短时间内有很多连接，需要增加它，换句话说，这是对到来的 TCP/IP 连接的侦听队列的大小。操作系统在这个队列大小上有它自己的限制。试图设定 back_log 高于操作系统的限制将是无效的。

当观察主机进程列表，发现大量 264084 | unauthenticated user | xxx.xxx.xxx.xxx | NULL | Connect | NULL | login | NULL 的待连接进程时，就要加大 back_log 的值。默认数值是 50，把它改为 500。

（2）interactive_timeout

服务器在关闭它前在一个交互连接上等待行动的秒数。一个交互的客户被定义为对 mysql_real_connect()使用 CLIENT_INTERACTIVE 选项的客户。默认数值是 28800，把它改为 7200。

（3）key_buffer_size

索引块是缓冲的并且被所有的线程共享。key_buffer_size 是用于索引块的缓冲区大小，增加它可得到更好处理的索引（对所有读和多重写），到能负担得起那样多。如果使它太大，系统将开始换页并且真的变慢。默认数值是 8388600（8MB），假如 MySQL 主机有 2GB 内存，可以把它改为 402649088（400MB）。

（4）max_connections

允许的同时客户的数量。增加该值能增加 mysqld 要求的文件描述符的数量。这个数字应该增加；否则，将经常看到 Too many connections 错误。默认数值是 100，把它改为 1024。

（5）record_buffer

每个进行一次顺序扫描的线程为其扫描的每张表分配这个大小的一个缓冲区。如果做很多顺序扫描，可能想要增加该值。默认数值是 131072（128KB），可以把它改为 16773120（16MB）。

（6）sort_buffer

每个需要进行排序的线程分配该大小的一个缓冲区。增加该值加速 ORDER BY 或 GROUP BY 操作。默认数值是 2097144（2MB），可以把它改为 16777208（16MB）。

（7）table_cache

为所有线程打开表的数量。增加该值能增加 mysqld 要求的文件描述符的数量。MySQL 对每个唯一打开的表需要两个文件描述符。默认数值是 64，可以把它改为 512。

（8）thread_cache_size

可以复用的保存在缓存中的线程的数量。如果有，新的线程从缓存中取得，当断开连接时如果有空间，客户的线程置在缓存中。如果有很多新的线程，为了提高性能可以设置这个变量值。通过比较 Connections 和 Threads_created 状态的变量，可以看到这个变量的作用。可以把它设置为 80。

（9）wait_timeout

服务器在关闭它之前在一个连接上等待行动的秒数。 默认数值是 28800，可以把它改为 7200。

注意，参数的调整可以通过修改 /etc/my.cnf 文件并重启 MySQL 实现。这是一个比较

谨慎的工作，上面的结果也仅仅是一些建议，可以根据主机的硬件情况（特别是内存大小）进一步修改。

12.5　小　　结

存储过程体中必须使用合法的 MySQL 语句，如果代码中包含 MySQL 扩充功能，那么代码将不能移植。

必须掌握新的 SQL 语句，Conditions and IF-THEN-ELSE 条件式和 IF-THEN-ELSE。有 3 种标准的循环方式：WHILE 循环、LOOP 循环以及 REPEAT 循环，还有一种不推荐使用的非标准的循环方式 GO TO。

必须掌握数据导入导出工具 mysqlimport 的语法和常用选项。

对于访问量很大的应用，MySQL 默认的系统参数很难保证 MySQL 运行得非常顺畅，需要通过 MySQL 关键系统参数进行性能优化。

第 13 章　MySQL 与 XML

本章主要介绍可扩展标记语言（Extensible Markup Language，XML）和扩展样式表语言（XML Document Transformation，XSLT）的语法、使用和应用，XML 是标准通用标记语言（Standard Generalized Markup Language，SGML）的一个子集，XSLT 是基于 XML 的样式表语言，用于转换 XML 文档。最后介绍 MySQL（5.1.5 版本以上）中的两个用于处理 XML 的新函数：ExtractValue()和 UpdateXML()。

13.1　XML

可扩展标记语言，缩写为 XML，描述了一类称为 XML 文件的数据对象，同时也描述了处理这些数据对象的计算机程序的动作。XML 是 SGML 针对特定应用领域的一个子集，或者说是 SGML 的一种受限形式。

SGML 是所有标记语言的母语言，HTML 和 XML 都派生自 SGML。SGML 定义了一种基本的语法，它允许创建自己的元素。要使用 SGML 描述一个特定的文档，必须创造一个恰当的元素集和文档结构。一个特定文档类型的通用元素集就是 SGML 应用程序。可以定义自己的 SGML 应用程序来描述所使用的特定文档类型，也可以定义一个标准体 SGML 应用程序来描述广泛使用的文档类型。例如 HTML，它是 1991 年开发的，用来描述 Web 页的 SGML 应用程序。

SGML 看起来似乎是用来描述 Web 文档的最佳扩展语言。但是，W3C 的成员认为 SGML 太复杂、太麻烦，无法有效地在 Web 上传递信息。SGML 所提供特性的灵活和冗余使得编写需要在 Web 浏览器中处理和显示 SGML 信息的软件变得困难。我们所需要的是专门为在 Web 上传递信息而设计的经过改进的 SGML 子集，即 XML。

【特别提示】 XML 由 XML 工作组（原来的 SGML 编辑审查委员会）开发，此工作组于 1996 年由 World Wide Web Consortium（W3C）组建。工作组由 Sun Microsystems 的 Jon Bosak 负责，同时，W3C 组织的 XML SIG（Special Interest Group，原来的 SGML 工作组）积极参与了 XML 工作组的工作。XML 不是 HTML 的替代品，XML 和 HTML 是两种不同用途的语言。XML 是被设计用来描述数据的，重点是“什么是数据，如何存放数据”。HTML 是被设计用来显示数据的，重点是“显示数据以及如何显示数据更好”。HTML 是与显示信息相关的，XML 则是与描述信息相关的。

13.1.1　XML 的 10 个设计目标

（1）XML 应在因特网上直接使用。

XML 主要是设计成用于网站上存储与传输信息。

（2）XML 应支持大量各类不同的应用。

XML 的主要目的是在网站上通过服务器与浏览器来传输信息，同时，XML 也被设计为供其他形式的软件使用。例如，XML 已被用来在金融软件间交换信息、分发与更新软件，以及被用来撰写声音 Script 以便能通过电话传输。

（3）XML 应与 SGML 兼容

XML 是 SGML 的一组特殊用途的子集。因此，SGML 软件工具可以轻易地处理 XML。

（4）应很容易编写处理 XML 文件的程序

XML 能够广泛地被应用的前提是浏览器与其他负责处理 XML 文件的程序应该很容易编写。事实上，由于编写负责处理 SGML 文件程序的不方便导致了 XML 子集的产生。

（5）XML 中的可选性功能的数目应该保持在最少数目，理想情况是零。

在 XML 中使用最少数目的可选性功能让开发人员在编写程序来处理 XML 文件时变得更加容易。在 SGML 中丰富的可选性功能就是 SGML 被认为无法定义网站文件的主要原因。

（6）XML 文件应易读且合理清楚。

XML 是为了成为使用者与应用程序之间信息交换的通用媒介而设计的。易读的特性让人们和某些特定的软件程序可以很容易编写及处理 XML 文件。XML 的易读性将 XML 从用于数据库与文字处理文件的大多数专有格式中区别出来。因为 XML 文件是以纯文字编写且拥有树状逻辑结构，所以人们可以轻易地阅读 XML 文件。

（7）XML 的设计应快速完成。

XML 只有在程序设计人员与使用者愿意接受它的情况下才能成为可实行的标准。因而该标准必须在用户开始接受其他替代标准之前完成，因为软件公司希望能快速地形成产品。

（8）XML 的设计应格式化且简洁化。

XML 的规格是用定义计算机语言的正规语言编写的，一般称之为 Extended Backus－Naur Form（EBNF）标签法。这种正规语言，虽然有时难以阅读，但它解决了意义不明确的问题，并在最后让编写 XML 文件与（尤其是）XML 处理软件变得更为容易，进而促进了 XML 被接受的可能性。

（9）XML 文件应易于建立。

为了让 XML 成为网站文件实际应用的统一语言，不只是 XML 的处理程序必须容易编写，而且 XML 文件本身也必须容易建立。

（10）XML 标签简化是最不重要的。

为了与目标（6）一致（XML 文件应易读且合理清楚），XML 标签不能太过简洁以免变得难以阅读。

13.1.2　XML 的语法简介

根据定义，XML 文件是合乎规范的 SGML 文件。XML 文件由称为实体的存储单元组成，实体可以包含已析数据或未析数据。已析数据由字符组成，其中一些字符组成字符数据，另一些字符组成标记。标记中包含了对文件存储格式和逻辑结构的描述。XML 提供了一种机制用于约束存储格式和逻辑结构。现在看一个 XML 文档的例子。

例 13-1　student.xml 文件内容如下：

```
<?xml version="1.0" encoding="gb2312"?>
<学生>
        <学号>072101004</学号>
        <姓名>方月红</姓名>
        <性别>女</性别>
        <年龄>20</年龄>
        <院系>信息系</院系>
</学生>
```

用 IE 打开 student.xml 效果，如图 13-1 所示。

```
<?xml version="1.0" encoding="gb2312" ?>
- <学生>
    <学号>072101004</学号>
    <姓名>方月红</姓名>
    <性别>女</性别>
    <年龄>20</年龄>
    <院系>信息系</院系>
  </学生>
```

（a）用 IE 浏览器浏览 student.xml

```
<?xml version="1.0" encoding="gb2312" ?>
+ <学生>
```

（b）单击根结点前的“-”时

图 13-1　用 IE 浏览器浏览 student.xml 的效果

文档的第 1 行是 XML 声明，定义此文档所遵循的 XML 标准的版本，在这个例子中是 1.0 版本的标准，使用的是 gb2312 简体中文字符集。第 2 行是根元素，说明这是一个学生的记录。第 3~7 行描述了根元素的 5 个子节点（学号、姓名、性别、年龄和院系）。最后一行表示根元素结束。从例子中可以看出，XML 文档使用了自描述的和简单的语法，编写读取和操作 XML 也相对容易。

（1）XML 元素命名

XML 元素命名必须遵守下面的规则：

- 元素的名字可以包含字母、数字和其他字符。
- 元素的名字不能以数字或者标点符号开头。
- 元素的名字不能以 XML（或者 xml、Xml、xMl...）开头。
- 元素的名字不能包含空格。

当然除了上述规则之外，还必须注意：

任何名字都可以使用，没有保留字（除了 XML），但是应该使元素的名字具有可读性，名字使用下划线是一个不错的选择。例如，<first_name>、<last_name>，尽量避免使用“-”，“.”，因为有可能引起混乱。

元素的名字可以很长，命名应该遵循简单易读的原则，例如，<book_title>是一个不错的名字，而<the_title_of_the_book>则显得罗嗦了。

XML 文档往往都对应着数据表，应该尽量让数据库中的字段的命名和相应的 XML 文档中的命名保持一致，这样可以方便数据变换。非英文/字符/字符串也可以作为 XML 元素的名字，如例 13-1。XML 元素命名中不要使用“:”，因为 XML 命名空间需要用到这个十分特殊的字符。

（2）所有的 XML 文档必须有一个结束标记

在 XML 文档中，忽略结束标记是不符合规定的。在 HTML 文档中，一些元素可以没有结束标记。下面的代码在 HTML 中是完全合法的：

```
<p>这是第一段
<p>这是第二段
```

但是在 XML 文档中必须要有结束标记，像下面的例子一样：

```
<p>这是第一段</p>
<p>第是第二段</p>
```

【特别提示】上面例子中的第一行并没有结束标记。这不是一个错误。因为 XML 声明并不是 XML 文档的一部分，它不是 XML 元素，也就不应该有结束标记。

（3）XML 标记大小写敏感

与 HTML 不一样，XML 标记是大小写敏感的。在 XML 中，标记<Letter>与标记<letter>是两个不同的标记。因此在 XML 文档中开始标记和结束标记的大小写必须保持一致。

试比较：

```
<Message>这是错误的</message>      //错误的
<message>这是正确的</message>      //正确的
```

（4）所有的 XML 元素必须合理嵌套包含

在 HTML 中，允许有一些不正确的包含，例如下面的代码可以被浏览器解析：

```
<b><i>This text is bold and italic</b></i>
```

在 XML 中所有元素必须正确地嵌套包含，上面的代码应该这样写：

```
<b><i>This text is bold and italic</i></b>
```

（5）所有的 XML 文档必须有一个根元素

XML 文档中的第一个元素就是根元素。所有 XML 文档都必须包含一个单独的标记来定义，所有其他元素都必须成对地在根元素中嵌套。XML 文档有且只能有一个根元素。所有的元素都可以有子元素，子元素必须正确地嵌套在父元素中，下面的代码可以形象地说明：

```
<root>
<child>
```

```
<subchild>......</subchild>
</child>
</root>
```

（6）属性值必须使用单引号或者双引号

在 XML 中，元素的属性值没有引号引着是不符合规定的。如同 HTML 一样，XML 元素同样也可以拥有属性，属性通常包含一些关于元素的额外信息。XML 元素的属性以名字/值成对地出现。XML 语法规范要求 XML 元素属性值必须用引号引着。请看下面的例子。

例 13-2　student1.xml 文件内容如下：

```
<?xml version="1.0" encoding="gb2312"?>
<学生表>
        <学生记录 记录号=001>
            <学号>072101004</学号>
            <姓名>方月红</姓名>
            <性别>女</性别>
            <年龄>20</年龄>
            <院系>信息系</院系>
        </学生记录>
        <学生记录 记录号="002">
            <学号>072102002</学号>
            <姓名>张三</姓名>
            <性别>男</性别>
            <年龄>19</年龄>
            <院系>物理系</院系>
        </学生记录>
</学生表>
```

当用 IE 打开 student.xml 时将会出错，错误之处是第一个学生记录的属性值没有用引号引着，第 3 行应改为“<学生记录　记录号='001'>”。

例 13-3　student2.xml 文件内容如下：

```
<?xml version="1.0" encoding="gb2312"?>
<学生表>
        <学生记录>
            <记录号>001</记录号>
            <学号>072101004</学号>
            <姓名>方月红</姓名>
        </学生记录>
</学生表>
```

比较例 13-2 与例 13-3，发现属性与子元素之间还可以互相转换。

【特别提示】有时应该为一个元素设计一个 ID 引用，通过这个 ID 可以引用存取特定的 XML 元素，就像 HTML 中的 name 和 id 属性一样。除了这种情况之外，应尽量避免使用属性，而应使用子元素代替。

（7）在 XML 中空白将被保留

在 XML 文档中，空白部分不会被解析器自动删除。这一点与 HTML 是不同的。在 HTML 中，这样的一句话：“Hello   my name is Tom"将会被显示成：“Hello my name is Tom”，因为 HTML 解析器会自动把句子中的空白部分去掉。

（8）XML 中回车或者换行均被转换为 LF（Line Feed，换行）

在 Windows 应用程序中，文本中的新行通常标识为 CR/LF（carriage return，回车）。在 UNIX 应用程序中，新行通常标识为 LF。还有一些应用程序只使用 CR 来表示一个新行。

（9）XML 中的注释

在 XML 中注释的语法基本上和 HTML 中的一样。用“<!--”开头，“-->”结尾。如：

<!--这是一个注释-->

注释可以在其他标记之外的文件中的任何位置出现。另外，它们可以在文件类型声明中文法允许的地方出现。出于兼容性考虑，字符串“--”（双连字符）不能在注释中出现。

（10）CDATA 段

CDATA 段可以出现在字符数据可以出现的任何地方，它们用于转义包含会被识别为标记的字符串的文本块。CDATA 段以字符串“<![CDATA[”开始，以字符串“]]>”结束。在 CDATA 内部的所有内容都会被解析器忽略。

如果文本包含了很多的“<”字符和“&”字符——就像程序代码一样，那么最好把它们都放到 CDATA 部件中。

例 13-4　source.xml 文件内容如下：

```
<?xml version="1.0" encoding="gb2312"?>
<script>
<![CDATA[
    function matchwo(a,b)
    {
        if (a < b && a < 0) then
        {
            return 1
        }
        else
        {
            return 0
        }
    }
]]>
</script>
```

在前面的例子中，所有在 CDATA 部件之间的文本都会被解析器忽略。

13.1.3　XML 的相关技术及应用简介

以下列出了一些很重要的 XML 相关技术：

（1）XHTML-可扩展 HTML（Extensible HTML）

XHTML 使用 XML 重新定义了 HTML 4.01 的语法，目前版本是 2.0。

（2）CSS-层叠样式表单（Cascading Style Sheets）

CSS 样式单可以为 XML 文档添加显示信息。

（3）XSL-可扩展样式单语言（Extensible Style Sheet Language）

XSL 由 3 部分组成：XML 文档转换（XML Document Transformation，又称 XSLT），模式匹配语法（a pattern matching syntax，又称 XPath），格式化对象（a formatting object interpretation，FO）。

（4）XSLT-XML 转换语言（XML Transformation）

XSLT 是一种比 CSS 强大很多的语言。它可以将 XML 文档转换成其他格式的文档。

（5）XPath-XML 匹配模式（XML Pattern Matching）

XPath 是一种用于标识 XML 文档各个部分的语言。这是一种为了 XSLT 和 XPointer 而设计出来的语言。

（6）XLink-XML 链接语言（XML Linking Language）

链接语言（The XML Linking Language，XLink），允许在不同的 XML 资源之间建立链接关系。

（7）XPointer-XML 指针语言（XML Pointer Language）

XML 指针语言（The XML Pointer Language，XPointer），标识 XML 文档的内部结构，例如元素、属性、内容等。

（8）DTD-文档类型定义（Document Type Definition）

DTD 主要用于定义编写 XML 文档所使用的元素。

（9）Namespaces-命名空间

XML 命名空间提供了一种可以把元素、属性、名字和 URL 地址引用相互关联的方法。

（10）XSD-XML 模式（XML Schema）

模式可以和 DTD 相互替代，并且功能更强大。模式使用 XML 格式编写，支持命名空间和数据类型。

（11）XDR-数据简化（XML Data Reduced）

XDR 是 XML 模式（XML Schema）的简化版本。

（12）DOM-文档对象模型（Document Object Model）

DOM 定义了 XML 文档的接口、属性和方法。

（13）XQL-XML 查询语言（XML Query Language）

XQL 为存放 XML 文档中的数据提供一种便捷的查询语言。

（14）SAX-XML 的简单 API（Simple API for XML）

SAX 是另一种读取和操作 XML 文档的编程接口（与 DOM 类似）。

XML 的灵活性意指可用于许多应用程序，如配置文件等，目前 XML 应用包括以下方面：

（1）存储数据

显而易见，XML 可用于存储数据。在以数据为中心的信息（如在某个数据库查找到的

数据）和以文档为中心的信息（如将数据存储在 XML 中，以便在不同的环境中显示出来）这两个方面，XML 都有自己的优势。

（2）Web 服务

Web 服务最初用于在 HTTP 上传递非 HTML 信息。它们如今已成为通过 Ajax 获取字段的基础，用于向 Web 站点、如今的面向服务体系结构（Service Oriented Architecture，SOA）和复杂的基于消息的应用程序添加交互性。XML 是 Web 服务领域不可或缺的一部分。Web 服务中的所有主流方法，如 SOAP、REST 甚至 XML-RPC，都是基于 XML 的。

（3）博客和其他数据联合

如今，XML 最普遍的应用是数据联合领域。数百万的 blogger 都在使用 RSS 提要订阅他们所喜爱的博客上的最新信息。而且商业利益已经发现通过英特网向各种设备（如 iPod，它也使用 XML）发布音频和视频的商机。

13.2　XSLT

XSL 指扩展样式表语言（EXtensible Stylesheet Language）。万维网联盟（W3C）开始发展 XSL 的原因是对基于 XML 的样式表语言的需求。可以比较 CSS 与 XSL 之间的关系。

（1）CSS 是 HTML 样式表

HTML 使用预先定义的标签，标签的意义很容易被理解。HTML 元素中的<table>元素定义表格，并且浏览器清楚如何显示它。向 HTML 元素添加样式是很容易的。通过 CSS，很容易告知浏览器用特定的字体或颜色显示一个元素。

（2）XSL 是 XML 样式表

XML 不使用预先定义的标签（可以使用任何喜欢的标签名），并且这些标签的意义并不都那么容易被理解。<table>元素在 XML 中意味着一张桌子还是一张表格，或是别的什么内容，浏览器不清楚如何显示它。

由此可见，HTML 通过按抽象概念（如段落、重点和编号列表）定义显示来实现设备独立性，而 XML 标记中使用的标记完全是用户定义的，其用意是这些标记应该与所关注的对象（如人、地点、价格和日期）相关。当然 XML 也可用 CSS，但 CSS 不能执行计算、重新整理或排序数据、组合多个源码中的数据或根据用户或会话的特征个性化显示的内容。

在 XSL 的开发过程中，发现在准备 XML 文档被显示的过程中执行的任务可以分成两个阶段：转换和格式化。转换是将一个 XML 文档转换成另一个 XML 文档的过程。格式化是将已转换的树状结构转换成两维图形表示法或可能是一维音频流的过程。XSLT 是为控制第一阶段“转换”而开发的语言。但实际上，大多数人现在使用 XSL 将 XML 文档转换成 HTML，并使用 HTML 浏览器作为格式化引擎。这是可行的，因为 HTML 实际上只是 XML 词汇表的一个示例，而 XSLT 可以使用任何 XML 词汇表作为其目标。

XSL 不仅仅是样式表语言，包括以下 3 个部分。

- ❑ XSLT（XML Document Transformation）：一种用于转换 XML 文档的语言。

- XPath（a pattern matching syntax）：一种用于在 XML 文档中导航的语言。
- XSL-FO（a formatting object interpretation）：一种用于格式化 XML 文档的语言。

简单地说，XSLT 把 XML 源树转换为 XML 结果树。

13.2.1　XPath 简介

XPath 是一门在 XML 文档中查找信息的语言，是 XSLT 标准中的主要元素。如果没有 XPath 方面的知识，就无法创建 XSLT 文档。XPath 可用来在 XML 文档中对元素和属性进行遍历。XPath 使用路径表达式来选取 XML 文档中的节点或者节点集。这些路径表达式和在常规的电脑文件系统中看到的表达式非常相似。XPath 含有超过 100 个内建的函数。这些函数用于字符串值、数值、日期和时间比较、节点和 QName 处理、序列处理、逻辑值等。可先了解一下有关 XPath 的术语。

（1）节点（Node）

在 XPath 中，有 7 种类型的节点：元素、属性、文本、命名空间、处理指令、注释以及根节点（又称之为文档节点）。如例 13-2 中，“<学生表>”是根节点，“<学生记录>”是元素，“记录号="002"”是属性。

（2）基本值（或称原子值，Atomic value）

如例 13-2 中的“方月红”及“002”都是基本值。

（3）条目（Item）

条目指基本值或者节点。

（4）节点关系

除根节点之外，每个元素以及属性都有一个父（Parent）。元素节点可有零个、一个或多个子节点（Children）。拥有相同的父的节点称之为同胞（Sibling），节点的父、父的父等称之为先辈（Ancestor），某个节点的子、子的子等称之为后代（Descendant）。

XPath 使用路径表达式来选取 XML 文档中的节点或节点集。节点是通过沿着路径（path）或者步（steps）来选取的。

例 13-5　student3.xml 文件内容如下：

```
<?xml version="1.0" encoding="gb2312"?>
<学生表>
        <学生记录 ID="001">
            <姓名 学号="072101004">方月红</姓名>
            <性别>女</性别>
            <年龄>20</年龄>
            <院系>信息系</院系>
        </学生记录>
        <学生记录 ID="002">
            <姓名 学号="072102002">张三</姓名>
            <性别>男</性别>
            <年龄>19</年龄>
            <院系>物理系</院系>
```

```
        </学生记录>
</学生表>
```

表 13-1 列出了常用路径表达式，可以对照表 13-2 用相应的表达式存取例 13-5 中的 XML 文件。

表 13-1　常用路径表达式

表　达　式	描　　述
节点名	选取此节点的所有子节点
/	从根节点选取
//	从匹配选择的当前节点选择文档的节点，而不考虑它们的位置
.	选取当前节点
..	选取当前节点的父节点
@	选取属性

表 13-2　例 13-5 路径表达式解析

路径表达式	结　　果
学生表	选取“学生表”元素的所有子节点
/学生表	选取根元素“学生表”
学生表/学生记录	选取所有属于“学生表”的子元素的“学生记录”元素
//学生记录	选取所有“学生记录”子元素，而不管它们在文档中的位置
学生表//学生记录	选择所有属于“学生表”元素后代的“学生记录”元素，而不管它们位于“学生表”之下的什么位置
//@ID	选取所有名为 ID 的属性

还可以通过谓语来查找某个特定的节点或者包含某个指定的值的节点。如例 13-5 所示，带有谓语的路径表达式有关举例如表 13-3 所示。

表 13-3　带有谓语的路径表达式举例

路径表达式	说　　明
/学生表/学生记录[1]	选取属于“学生表”子元素的第一个“学生记录”元素
/学生表/学生记录[last()]	选取属于“学生表”子元素的最后一个“学生记录”元素
/学生表/学生记录[last()−1]	选取属于“学生表”子元素的倒数第二个“学生记录”元素
/学生表/学生记录[position()<3]	选取最前面的两个属于“学生表”元素的子元素的“学生记录”元素
//姓名[@学号]	选取所有拥有名为“学号”的属性的“姓名”元素
//姓名[@='072101004']	选取所有“姓名”元素，且这些元素拥有值为“072101004”的“学号属性”
/学生表/学生记录[年龄>19]|/学生表/学生记录[性别='女']	选取所有“学生表”元素的“学生记录”元素，其中的“年龄”元素的值需大于 19，“性别”元素的值为“女”
/学生表/学生记录[年龄=20]/院系	选取所有“学生表”元素中的“学生记录”元素的“院系”元素，且其中的“年龄”元素的值等于 20
child::学生表	选取所有属于当前节点的子元素的“学生表”节点
child::*	选取当前节点的所有子元素

续表

路径表达式	说　　明
attribute::ID	选取当前节点的“ID”属性
child::text()	选取当前节点的所有文本子节点
child::node()	选取当前节点的所有子节点
descendant::学生表	选取当前节点的所有“学生表”后代
ancestor::学生记录	选择当前节点的所有“学生表”先辈
ancestor-or-self::学生记录	选取当前节点的所有“学生记录”先辈以及当前节点
child::* child::学生表	选取当前节点的所有“学生表”的孙

13.2.2　XSLT-转换

通过对 XML 文档进行语法分析来生成输入树状结构，而输出树状结构通常被串行化到另一个 XML 文档中。但 XSLT 处理器本身操作的是树状结构，而不是 XML 字符流，如图 13-2 所示。

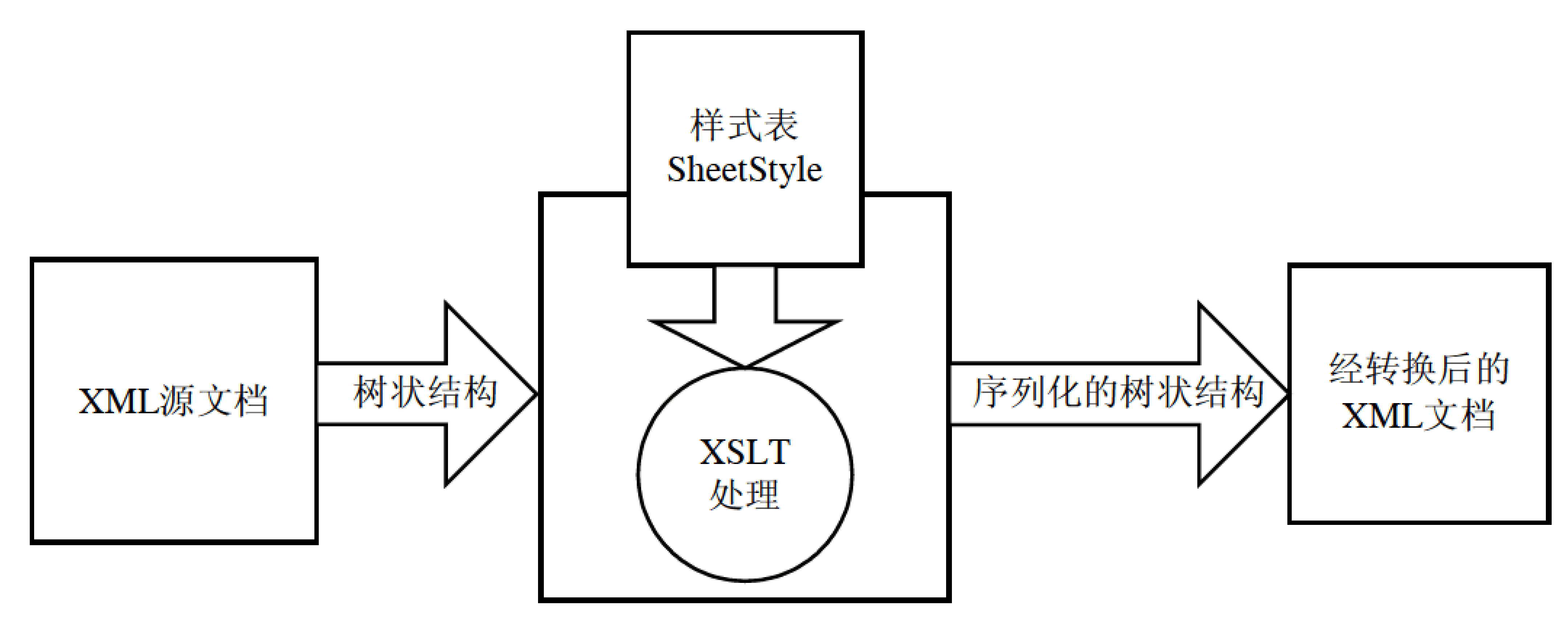


图 13-2　XSLT 转换过程

XSLT 以传统语言（如 Lisp、Haskell 和 Scheme）中的功能性编程概念为基础。样式表由模板组成，这些模板基本上是单一功能——每个模板将输出树的一部分定义成一部分输入树的功能，并且不产生副作用。使用无副作用的规则受到严格控制（除了转义成用类似 Java 的语言编写的外部代码）。XSLT 语言允许定义变量，但不允许现有变量更改它的值，即没有赋值语句。

例 13-6　student.xsl 文件内容如下：

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
```

```
    <h2>学生基本信息表</h2>
    <table border='1'>
    <tr>
      <th align="left">姓名</th>
      <th align="left">年龄</th>
    </tr>
    <xsl:for-each select="学生表/学生记录">
    <tr>
      <td><xsl:value-of select="姓名"/></td>
      <td><xsl:value-of select="年龄"/></td>
    </tr>
    </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

可以理解一下以上代码。由于 XSL 样式表本身也是一个 XML 文档，因此它总是由 XML 声明起始：

```
<?xml version="1.0" encoding="gb2312"?>
```

根据 W3C 的 XSLT 标准，声明一个 XSL 样式表的正确方法是：

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

【特别提示】把文档声明为一个 XSL 样式表的根元素是<xsl:stylesheet>或<xsl:transform>，即也可以这样声明：

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

如需访问 XSLT 的元素、属性以及特性，必须在文档顶端声明 XSLT 命名空间“xmlns:xsl="http://www.w3.org/1999/XSL/Transform”指向官方的 W3C XSLT 命名空间。如果使用此命名空间，就必须包含属性 version="1.0"。

<xsl:template>元素用于构建模板。match 属性用于关联 XML 元素和模板。match 属性也可用来为整个文档定义模板。match 属性的值是 XPath 表达式（例子中 match="/"定义整个文档）。介于<xsl:template>与</xsl:template>之间是具体的转换过程。最后两行定义了模板的结尾，及样式表的结尾。

<xsl:value-of>元素用于提取某个选定节点的值，并把值添加到转换的输出流中，select 属性的值是一个 XPath 表达式。如例子中<xsl:value-of select="姓名"/>表示把“姓名”节点的值添加到当前输出流中。

<xsl:for-each>元素可用于选取某个指定节点集的每个 XML 元素，select 属性的值也是一个 XPath 表达式。如例子中<xsl:for-each select="学生表/学生记录">表示选取所有属于“学生表”的“学生记录”。

如果要使 student3.xml 能按照 student.xsl 格式来转换显示，则还需要在 student.xml 文件中第二行加入：

```
<?xml-stylesheet type="text/xsl" href="student.xsl"?>
```

以此说明本 XML 文档通过 student.xsl 转换，效果如图 13-3（a）所示。

通过在 <xsl:for-each> 元素中添加一个选择属性的判别式，也可以过滤由 XML 文件输出的结果。例如把例 13-6 处的<xsl:for-each>语句改为如下：

```
<xsl:for-each select="学生表/学生记录[年龄=20]">
```

则显示的效果如图 13-3（b）所示。

【特别提示】 合法的过滤运算符有 4 种：=（等于）、!=（不等于）、<（小于）、>（大于）。

由于 HTML 语言是用“<”、“>”符号进行特殊标记，这里不能直接在 xsl 文档中使用“<”和“>”。

还可以利用<xsl:sort>元素对结果进行排序。例如，在例 13-6 中语句<xsl: for-each select = "学生表/学生记录">后加入<xsl:sort select="年龄"/>，则显示效果如图 13-3（c）所示。

学生基本信息表

姓名	年龄
方月红	20
张三	19

（a）

学生基本信息表

姓名	年龄
方月红	20

（b）

学生基本信息表

姓名	年龄
张三	19
方月红	20

（c）

图 13-3　用 IE 打开经 student.xsl 转换后的 student.xml

如需放置针对 XML 文件内容的条件测试，可向 XSL 文档添加 <xsl:if> 元素。例如：

```
<xsl:for-each select="学生表/学生记录[年龄!=21]">
  <xsl:if test="年龄 &gt; 19">
  <tr>
    <td><xsl:value-of select="姓名"/></td>
    <td><xsl:value-of select="年龄"/></td>
  </tr>
  </xsl:if>
</xsl:for-each>
```

以上代码实现了显示年龄大于 19 的学生记录的效果。

XSLT<xsl:choose>元素用于结合<xsl:when>和<xsl:otherwise>来表达多重条件测试。

例 13-7　student2.xsl 文件内容如下：

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
```

```
<body>
    <h2>学生基本信息表</h2>
    <table border='1'>
        <tr>
            <th align="left">姓名</th>
            <th align="left">年龄</th>
        </tr>
        <xsl:for-each select="学生表/学生记录[年龄!=21]">
        <tr>
            <xsl:choose>
                <xsl:when test="年龄 &lt; 19">
                    <td bgcolor="#ff00ff">
                    <xsl:value-of select="姓名"/></td>
                </xsl:when>
                <xsl:when test="年龄 &gt; 19 ">
                    <td bgcolor="#0000ff">
                    <xsl:value-of select="姓名"/></td>
                </xsl:when>
                <xsl:otherwise>
                    <td><xsl:value-of select="姓名"/></td>
                </xsl:otherwise>
              </xsl:choose>
              <td><xsl:value-of select="年龄"/></td>
              </tr>
            </xsl:for-each>
        </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

运行后的效果如图 13-4 所示。

学生基本信息表

姓名	年龄
方月红	20
张三	19

图 13-4　用 IE 打开经 student2.xsl 文件转换后的 student.xml 运行时的效果

13.3　XML、MySQL 的结合运用

MySQL 5.1.5 版本中添加了对 XML 文档进行查询和修改的函数，分别是 ExtractValue() 和 UpdateXML()。

13.3.1　ExtractValue()函数

语法：EXTRACTVALUE (XML_document, XPath_string)

- 第一个参数：XML_document 是 String 格式，为 XML 文档对象的名称。
- 第二个参数：XPath_string（Xpath 格式的字符串）。

作用：从目标 XML 中返回包含所查询值的字符串。

首先在 MySQL 中建立一个用于测试 XML 的表 myXML，并插入两条记录：

```
mysql> CREATE TABLE myXML(doc VARCHAR(150));
INSERT INTO myXML VALUES
('
        <stuRecord>
            <sname sid="072101004">方月红</sname>
            <ssex>女</ssex>
            <sage>20</sage>
            <sdept>信息系</sdept>
        </stuRecord>
');
INSERT INTO myXML VALUES
('
        <stuRecord>
            <sname sid="072102002">张三</sname>
            <ssex>男</ssex>
            <sage>19</sage>
            <sdept>物理系</sdept>
        </stuRecord>
');
```

SELECT ExtractValue(doc,"/stuRecord/sname")FROM myXML；现在可以利用 ExtractValue()函数对记录内容中的 XML 文档进行读取。

例 13-8　读取 myXML 表 doc 字段中姓名 sname。

```
mysql> SELECT ExtractValue(doc,"/stuRecord/sname") FROM myXML;
```

返回结果如下：

```
+--------------------------------------+
| ExtractValue(doc,"/stuRecord/sname") |
+--------------------------------------+
| 方月红                               |
| 张三                                 |
+--------------------------------------+
2 rows in set
```

可以看到，EXTRACTVALUE()函数将/stuRecord/sname 节点中的值取出来，并通过 SELECT 返回。因此需要简单地查找 XML 文档中的值，只要在 XPath_string 参数中指定好层次和节点即可。

例 13-9　读取 myXML 表 doc 字段中姓名 sname 和年龄 sage 的信息。

```
mysql>SELECT ExtractValue(doc,"//sage|//sname") FROM myXML;
```

返回结果如下：

```
+------------------------------------+
| ExtractValue(doc,"//sage|//sname") |
+------------------------------------+
| 方月红 20                          |
| 张三 19                            |
+------------------------------------+
2 rows in set
```

如果查询前并不知道层次关系，也可以使用通配符进行层次的匹配，不过当 XML 文档比较大时查找速度会很慢。

例 13-10　读取 myXML 表 doc 字段中所有院系 sdept 信息。

```
mysql>SELECT EXTRACTVALUE(doc,'/*/sdept') FROM myXML;
```

返回结果如下：

```
+-----------------------------+
| EXTRACTVALUE(doc,'/*/sdept') |
+-----------------------------+
| 信息系                       |
| 物理系                       |
+-----------------------------+
2 rows in set
```

还可以用表 13-3 中的相关谓词进行查询，例如使用/stuRecord/child::语句可以找到 stuRecord 节点下的首个节点。

例 13-11　读取 myXML 表 doc 字段中所有学生记录子节点信息。

```
mysql> SELECT ExtractValue(doc,'/stuRecord/child::*') FROM myXML;
```

返回结果如下：

```
+-----------------------------------------+
| extractValue(doc,'/stuRecord/child::*') |
+-----------------------------------------+
| 方月红 女 20 信息系                      |
| 张三 男 19 物理系                        |
+-----------------------------------------+
2 rows in set
```

例 13-12　读取 myXML 表 doc 字段中姓名 sname 的值为“方月红”的信息。

```
mysql> SELECT ExtractValue(doc,'/stuRecord/sname[self:text()="方月红"]')
FROM myXML;
```

返回结果如下：

```
+-------------------------------------------------------------+
| extractValue(doc,'/stuRecord/sname[self:text()="方月红"]') |
+-------------------------------------------------------------+
| 方月红                                                      |
|                                                             |
+-------------------------------------------------------------+
2 rows in set
```

ExtractValue()函数可以使用在任何一个允许使用表达式的语句中。

13.3.2　UpdateXML()函数

语法：UPDATEXML (XML_document, XPath_string, new_value);

- 第一个参数：XML_document 是 String 格式，为 XML 文档对象的名称。
- 第二个参数：XPath_string（Xpath 格式的字符串）。
- 第三个参数：new_value，String 格式，替换查找到的符合条件的数据。

作用：改变文档中符合条件的节点的值。

例 13-13　试运行以下语句：

```
mysql> SELECT UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>fff</e>');
```

返回结果如下：

```
+-----------------------------------------------------------+
| UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>fff</e>') |
+-----------------------------------------------------------+
| <e>fff</e>                                                |
+-----------------------------------------------------------+
1 row in set
```

```
mysql> SELECT UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>fff</e>');
```

返回结果如下：

```
+-----------------------------------------------------------+
| UpdateXML('<a><b>ccc</b><d></d></a>', '//b', '<e>fff</e>') |
+-----------------------------------------------------------+
| <a><e>fff</e><d></d></a>                                  |
+-----------------------------------------------------------+
1 row in set
```

UpdateXML()函数的前两个参数用法以及写法与 ExtractValue 是一样的，因为这里需要查找到符合条件的数据节点。第三个参数就是需要替换节点字符串的值。例 13-14 中返回值是整个改变后的 XML 文档。

例 13-14　修改 myXML 表 doc 字段中年龄 sage 显示值为“22”。

```
mysql> SELECT UpdateXML(doc,'/stuRecord/sage','22') FROM myXML;
```

返回结果如下：

```
+--------------------------------------------------------------
| UpdateXML(doc,'/stuRecord/sage','22')
+--------------------------------------------------------------
|
                <stuRecord>
                        <sname sid="072101004">方月红</sname>
                        <ssex>女</ssex>
                        22
                        <sdept>信息系</sdept>
                </stuRecord>
  |
|
                <stuRecord>
                        <sname sid="072102002">张三</sname>
                        <ssex>男</ssex>
                        22
                        <sdept>物理系</sdept>
                </stuRecord>
    |
+--------------------------------------------------------------
2 rows in set
```

【特别提示】由于使用的是 SELECT 语句，因此并没有对真正的数据进行修改，而是在内存中将取出的数据进行修改，然后返回给用户。

如果需要彻底地修改文档内容，可以使用下面语句：

```
UPDATE myXML SET doc = UpdateXML(doc,'/stuRecord/sage','22');
SELECT doc FROM myxml;
```

显示结果如下：

```
+----------------------------------------------------------------
| doc
+----------------------------------------------------------------
|
                <stuRecord>
                        <sname sid="072101004">方月红</sname>
                        <ssex>女</ssex>
                        22
                        <sdept>信息系</sdept>
                </stuRecord>
 |
|
                <stuRecord>
                        <sname sid="072102002">张三</sname>
                        <ssex>男</ssex>
                        22
                        <sdept>物理系</sdept>
                </stuRecord>
   |
+----------------------------------------------------------------
2 rows in set
```

但要注意的是，以上语句执行之后，发现原来“<sage>20</sage>”变成了“22”，即把结点名称也更改了，显然，不符合我们的原意。因此，可以改为运行以下语句：

```
UPDATE myXML SET doc = UpdateXML(doc,'/stuRecord/sage','<sage>22</sage>');
```

也可以把 ExtractValue 与 UpdateXML 合起来写：

```
mysql>SELECT ExtractValue( UpdateXML( doc, '/stuRecord/sage' , '<sage>22</
sage>' ) , '/stuRecord/sage' )  FROM  myXML;
```

返回结果如下：

```
+------------------------------------------------------------------------------------+
| extractvalue(UpdateXML(doc,'/stuRecord/sage','<sage>22</sage>'),'/stuRecord/sage') |
+------------------------------------------------------------------------------------+
| 22                                                                                 |
| 22                                                                                 |
+------------------------------------------------------------------------------------+
2 rows in set
```

13.4　小　　结

可扩展标记语言 XML 描述了一类称为 XML 文件的数据对象，具有非常明显的优点。

XSL 包括 XSLT、XPath 和 XSL-FO 3 部分。XSLT 的主要用途就是数据转换应用。可以把有用数据从 XML 文档中提取出来，而且可以利用各种不同的 XSLT 模板处理，输出各种不同需求的文档。

由于以 XML 为基础的电子商务广泛普及，XSLT 作为数据转换的角色也越来越重要。例如，直接将电视新闻的数据格式转换成报纸新闻需要的数据格式，将股票数据直接转换成图片显示在网页上，对 EDI（电子数据交换）数据进行统计、排序等。

《程序员突击——MySQL 原理与 Web 系统开发》
读者建议反馈表

1、姓名__________ 2、性别________ 3、年龄_________ 4、电话__________________
5、单位____ 6、职务/职称 ____________
7、通信地址 ____邮编 ___________________
8、电子信箱 ____________________________________单位网站 ___________________
9、您的文化程度： □中专以上 □大专 □本科 □研究生以上
10、您所学专业： □通信电子 □计算机类 □机电控制 □数学类 □其他
11、您所在行业： □商业网站 □硬件开发 □邮政银行 □软件企业 □系统集成 □服务行业 □科研院校 □政府机关 □网络通信 □制造业
12、您的工作性质： □设计开发 □大学教学 □普通培训 □学生
13、您使用计算机在：□办公室 □实验室 □网吧 □宿舍和家 □笔记本电脑
14、您每季度买书在：□五十元内 □一百元内 □二百元内 □可以报销
15、您购买本书在： □新华书店 □校园书店 □科技书店 □网站 □其他
16、您使用编程语言：□C/C#/C++ □Java □Delphi □VB □其他
17、您使用数据库为：□Oracle □DB2 □SQL Server □Sybase □其他
18、您开发平台为： □.Net □JavaOne □UNIX □Linux □其他
19、您认为本书作者应该创作哪些其他书籍，如何写

____

____

____

20、您认为市面上类似书籍的特点有哪些

____

____

____

21、您对本书的建议和意见

____

____

22、您今后需要那些本类的书籍

____

____

读者咨询方式

北京清华大学出版社第六事业部 邮编：100084
电话：010-62788951/62791976 转 288 传真：010-62788903
信箱：thuxj@126.com

有关本书的建议和意见或邮购本丛书请按照以下方式联系：

地址：北京清华大学校内出版社白楼二楼 邮编：100084
电话：010-62770381 传真：010-62788903
公司网址：www.tup.com.cn 电子信箱：thuxj@126.com